# THE OPTIONS DESIGN EXPLORATION SYSTEM

Reference Manual and User Guide - Version B3.1

A.J. KEANE

# OPTIONS

This manual and user guide gives details of the **OPTIONS** design exploration package. It contains a general introduction to DES's, background information, worked examples, textural and graphical output. The appendices detail the data-base manipulation package used by the system, optimization strategies, terminal logs taken from the examples and references in the general literature. Additionally provided as appendices, are the source codes for the examples studied.

## Table of Contents

# List of Figures

Figure are placed at the ends of relevant chapters.

# CHAPTER 1

# Introduction

## 1.1. Design Exploration Systems

Design exploration systems (DES's) address the needs of engineers responsible for establishing critical design parameters, usually during the early stages of the design process. Such systems aim to provide a controlled framework for studying the effects of parameter choices on the traditional programs available to the designer for dealing with geometric decisions, stress analysis, performance estimation, etc. They have started to become available over the last five years with the advent of relatively cheap and powerful computational facilities and sophisticated optimization and exploration techniques. The increasing cost of professional manpower and pressure to gain maximal performance from designs makes it reasonable to expect that DES's will grow in use and influence.

The heart of a good DES is usually a design optimization facility; **OPTIONS** includes a particularly powerful and varied suite of optimizers covering a wide range of methods. The purpose of these tools is to modify a design in such a way as to increase (or reduce) some measure of merit for the design while satisfying various constraints and other requirements, i.e., to improve the design in some specific area while meeting other restrictions. In the jargon of optimization the measure of merit to be minimized or maximized is usually called the *objective function*, the restrictions on parameters a *constraint vector*, while the group of variables that characterize a particular solution is commonly known as a *trial* or *design vector*. It is the trial vector that is changed at the beginning of each step of an optimization search. The objective function is usually a measure of cost or performance, or any parameter or combination of parameters where maximization (or minimization) is desirable. To achieve an optimum the optimizer modifies the variables selected by the user to form the trial vector according to the strategy in use. These variations are usually carried out within pre-established upper and lower limits set in the design data-base. When seeking constrained optima, which is the normal case, the optimizer additionally tries to fulfill separate constraint requirements and keeps records of constraint violations.

Although good optimization facilities are crucial to the performance of a DES the user must also be presented with tools that allow for manually controlled interaction and search: often a mixture of user controlled and optimization searches will be the preferred mode of use for a good system. **OPTIONS** provides such facilities allowing fully manual design work to be carried out in a flexible fashion. This is achieved by allowing evaluation of the objective function at any point under manual control. This causes the design software to be exercised once and the results stored in the design data-base. Additionally, **OPTIONS** allows the objective function to be systematically mapped by varying parameters through user selected ranges of values, holding other variables constant. The results of such mappings can then be plotted as graphs, contour plots or iso-metric surfaces to aid understanding. These three distinct modes of operation form the key to the **OPTIONS** system.

In addition to providing a stand-alone design exploration system, the optimization routines with the **OPTIONS** suite may also be accessed directly via two simple subroutines that can be incorporated directly into a user's own code. This allows the user to access the whole range of search methods through whatever interface the user's design process is normal handled by.

## 1.2. Background to the OPTIONS System

DES systems have been of interest to the designers of the **OPTIONS** system for at least fifteen years. They have experience of systems in teaching, research and governmental environments where they have been closely involved in the development of a number of different systems and a wide range of techniques. **OPTIONS**, which is the most recently developed of these, draws on this experience and provides a comprehensive, powerful and easy to use tool. This document describes the system and gives a number of worked examples, together with a series of useful appendices describing some of the underlying techniques. It has been mainly produced under the UNIX operating system on Sun workstations (using XView), but is written in ANSI 77 Fortran allowing it to be run on many common computers. Graphical output, making use of the widely known Simpleplot[1] library, is an option for which a separate license must be obtained if required. Interfaces to the OPTIVAR[2], Schwefel[3], Numerical Recipes[4] and NAg[5] optimization libraries are also available.

# CHAPTER 2

## OPTIONS Structure

### 2.1. Overview

The functional structure of the **OPTIONS** design system is illustrated in Figure 2.1. This structure is built from a *User Interface Manager*, a *Data-base Handler*, an *Optimizer Suite*, between two and four user supplied *Design Theory Modules*, plus some *Auxiliary Routines*, all linked to a central *Design Data-base*. The underlying approach has been to develop a flexible overall framework together with a range of custom built and third party supplied optimization and exploration modules. As with all such systems, the user must supply the problem specific code that describes the design area being studied. This is linked into the **OPTIONS** system either using the system linker or via system calls at run time. The main component modules currently comprise:

GENDAT for design data-base and menu handling;

OPTCTL for function evaluation, mapping and optimization control;

SGRAPH for general purpose graphical output;

TRNSFM for carrying out simple manipulations and calculations on numeric data;

OPTPRE to carry out any problem specific design pre-processing;

OPTFUN to evaluate the designer's chosen objective(s);

OPTCON to evaluate any constraints applying to the design and

OPTPST to carry out any problem specific design post-processing.

In addition, there are two further modules that may be used to provide direct subroutine access to the search methods within the **OPTIONS** suite:

OPTDBS to construct an internal **OPTIONS** data-base directly from the parameters supplied by the user's own code and

OPTNS to run a search on the user's problem using one of the methods available in the **OPTIONS** suite.

The modules OPTPRE, OPTFUN, OPTCON and OPTPST are normally user supplied and can range from the very simple to the highly complex. They may well incorporate many other routines and programs and evaluate a whole range of possible objectives and constraints which can then be studied as the user desires. The subdivision into four routines allows for the most efficient coding of the user problem, since only those pieces of code needed for each specific aspect need then be run when design changes are made. In particular, OPTPRE deals with calculations that will only be needed to help set up a problem once the user has decided on the fixed parameters. Similarly OPTPST deals with final post-processing carried out after all design variables have been chosen. Both OPTPRE and OPTPST can be omited if desired. OPTFUN deals with evaluating the objective function and OPTCON the constraints. Thus, for example, when an optimizer is simply searching for a feasible region it need only call OPTCON, using OPTFUN just when it has found points that are worth further study: OPTPRE and OPTPST being called only once during the search (at the beginning and end).

If the user's problem is such that the calculation of constraints and objective function cannot be seperated into two distinct parts the user should provide this combined code in a single subroutine called OPTFNC instead of OPTCON and OPTFUN. Internal caching ensures that the one, combined, routine will not be called twice with the same parameter set to avoid wasting computational effort.

It is also important to remember that the **OPTIONS** package uses the fortran logical unit numbers 10 and 11 for a number of purposes and therefore these LUNs should be avoided in user written code.

Before passing on to consider specific examples, a brief description of the main in-built elements is given in the following sections.

## 2.2. Data-base Handler (GENDAT)

As part of the development of the **OPTIONS** suite, the GENDAT design data-base handler has been produced. This takes the form of a library of Fortran and C callable functions that may be readily incorporated into new programs. The primary function of this package is to provide the means of controlling and accessing a design data-base. The requirement for flexibility in this capability means that, in addition to providing the means for storing and retrieving data, it also provides a powerful, program definable, user interface. This interface currently makes use of the Sun XVeiw package, which is a mouse driven, X-windows package, available on many modern workstations. It may, however, be used on any simple terminal, albeit in a less sophisticated fashion. Appendices A and B give further details of the GENDAT package, being the user guide and documentation file, respectively.

When the **OPTIONS** package is run, the GENDAT interface is entered and it initiates a menu driven interactive process with the user. It is from this level that all subsequent routines are accessed and controlled. Using the package a design description file can be recalled from disk and calculated results sent back. It additionally incorporates features that allow for a problem specific data-base to be set up, the input of values for the various parameters of interest and the selection of suitable constraints and objectives together with the quantities to be varied during mapping or optimization. It also allows the creation of formatted output files for subsequent printing. The design data-base maintained by this system contains the names, values, types, units, meanings, etc., for the various quantities making up the current design description and is thus self contained and self documenting. The variables within the data-base can be manipulated individually or in groups, enabling the user to have great flexibility and ease of control. Finally, it is through the GENDAT interface that commands are given to initiate design explorations using the functions of **OPTIONS** that are defined in OPTCTL. The major menus used, their specific functions and inter-relations are illustrated in Figure 2.3.

## 2.3. Function Evaluator, Mapper and Optimizer Control (OPTCTL)

A number of different evaluation, mapping and optimization strategies are available within the system to allow the effects of design choices to be studied. It is emphasized that full multi-parameter optimization is only one of the possible ways of using the package to explore design decisions.

Function evaluation causes the system to check the design data-base for self consistency and then to make a single pass through the user supplied design routines. These may be set up so that all the user's code is run or merely sufficient to calculate the current functions of interest. This latter feature is useful where long run times are involved and a variety of different objectives and constraints may need to be studied. To make use of this facility the user supplied programs are coded to evaluate all the variables likely to be of interest, in the most efficient order, interspersed with calls to a monitoring routine. The package is then capable of detecting when sufficient calculations have been performed to deal with the variables of interest during any particular study, without the overhead of repeated edits and relinks. The package is also able to detect if the user supplied code does not, in fact, evaluate the required objective function and constraints, which provides a useful safeguard against coding errors.

Function mapping allows one or two user selected quantities to be systematically varied while holding all other inputs constant. The range of values considered can be set automatically or chosen by the user, either with uniform spacings or via a user specified list of values. The results are typically presented as a contour map but sometimes isometric projections are more useful; both kinds of plots may be produced (with just one variable, simple graphs, which represent cross-sections through the optimization surface, are produced). This capability has two principal uses: first, it allows the user to gain insight into how rapidly the objective function alters as the design changes, allowing sensitivity checks to be made. Secondly, it allows the boundaries of feasible designs to be explored. Such boundaries arise either because constraints are violated or because the relevant combinations of parameters do not give rise to valid designs. Both checks can be important when developing new ideas and in some cases all that a design study requires is the systematic preparation of a number of such maps. This may be the case if they give a complete description of the design space or, alternatively, if the user is sufficiently confident in the choice of key parameters to render optimization studies unwarranted. However, when many quantities are possible candidates for variation and these have subtle interactions, full optimization becomes necessary, perhaps followed by further mapping around a number of interesting design configurations.

The optimizers within the software are either custom built or drawn from the OPTIVAR[2], Schwefel[3], Numerical Recipes[4] or NAg[5] packages (OPTIVAR is a public domain library of general purpose optimization routines collated by Siddall, Schwefel provides a collection of optimization routines with his book, the Numerical Recipes book contains a public domain library of general mathematical software that includes optimization facilities while NAg provide a commercial library of general mathematical software that also includes optimization facilities). The package based optimizers give a fairly comprehensive coverage of traditional, slope based methods while the custom built routines deal with modern stochastic techniques. These currently include Genetic Algorithm[6], Simulated Annealing[7], Evolutionary Programming[8], Dynamic Hill Climbing[9], Evolutionary Strategy[10], Design of Experiments[11], Bit Climbing[12] and Population Based Incremental Learning[13] methods. The package also provides a simple to use interface that allows up to two user designed optimization methods to be merged into the package.

Many design problems are best dealt with by applying a combination of different optimizers to deal with different aspects, with the modern stochastic methods often being most useful for wide ranging searches over many dimensions while the traditional methods are used for fine tuning near the desired optimum configuration. The optimization techniques available within the **OPTIONS** package are further described in Appendix D for convenience.

The OPTIVAR suite contains seven main methods of unconstrained nonlinear optimization with four associated penalty functions, together with two constrained nonlinear methods. These represent a wide range of well established heuristic methods which are fully described in the reference.

The library provided by Schwefel contains 11 classical slope driven and two modern stochastic methods. The classical optimizers are similar to those in the OPTIVAR package but are sufficiently differently implemented to often give different results from those methods. The stochastic methods are two membered and multi-membered evolution strategies.

The optimization routines in the the Numerical Recipes cookbook are again classical slope driven methods but additionally allow for codes where the user can provide the local gradient of the objective function more efficiently than by the finite differencing used by the other classical techniques in the package.

The optimizer provided by NAg is called E04UCF and can handle linear and non-linear constrained and unconstrained problems. It uses uses a sequential quadratic programming (SQP) algorithm which is fully described in the manuals supplied by NAg. It is essentially, however, a slope based traditional method.

The stochastic techniques are based on ideas culled from the literature plus development work carried out specifically for the **OPTIONS** package. They all work by carrying out stochastic variations on the existing design over a sequence of generations which evolve towards the optimal design. Their great advantage over traditional methods is that their performance is not greatly affected by the dimensions of the search space, although they can be rather imprecise and also slow on low dimensional problems. They are also more robust and less often get stuck in local optima than more traditional methods.

### 2.4. Graphical Output (SGRAPH)

The routine SGRAPH allows access to a whole range of general purpose plotting and contouring software based on the Simpleplot[1] graphics package. A number of custom built graphics device drivers are supplied to work with this package so that the resulting output is neatly integrated into the main user interface. Multiple curves can be placed on each plot with multiple plots on each page. Scales can be linear, logarithmic, polar or probability based and various types of smoothing and curve fitting are available. Hardcopy output can be generated via postscript or HPGL and this can be suitably customized for inclusion in final reports, etc. Appendix C gives the user guide for this part of the package, which may also be used in stand-alone fashion via the program "graph" which is supplied with package.

### 2.5. Data-base Numeric Manipulator (TRNSFM)

When working with design data-bases it is often desirable to make simple calculations on numeric data to produce new variables or modified versions of them, e.g., to convert frequency points from rad/s to Hz. or to add two variables together. TRNSFM provides a basic library of simple mathematical functions that can be used for this purpose, thus reducing the need to transfer data to other packages such as spreadsheets.

### 2.6. Internal Data-base Construction Sub-routine (OPTDBS)

When using the direct subroutine interface to the optimizers this routine is called by the user to establish the internal **OPTIONS** design data-base needed to link the user's problems to the **OPTIONS** search engines.

### 2.7. Optimizer Call Sub-routine (OPTNS)

When using the direct subroutine interface to the optimizers this routine is called to run a search engine on the user's problem, as previously specified in the call to OPTDBS. The search method to be used and start point are set as parameters to this routine and results returned so that repeated calls to the routine may be used to construct a composite search strategy for the problem in hand.

If using this approach to call the **OPTIONS** search engines directly much of this manual may not appear directly relevant, since it is primarily geared to those using the **OPTIONS** user interface and data-bases on an interactive basis. Nonetheless, it is still worth scanning chapters three to six and eight before turning to chapter nine as these sections contain much relevant material. Chapter nine then gives a detailed introduction to this approach to handling design problems. Experienced users of **OPTIONS** who are planning to use OPTDBS and OPTNS will probably just need to refer to chapter nine directly.

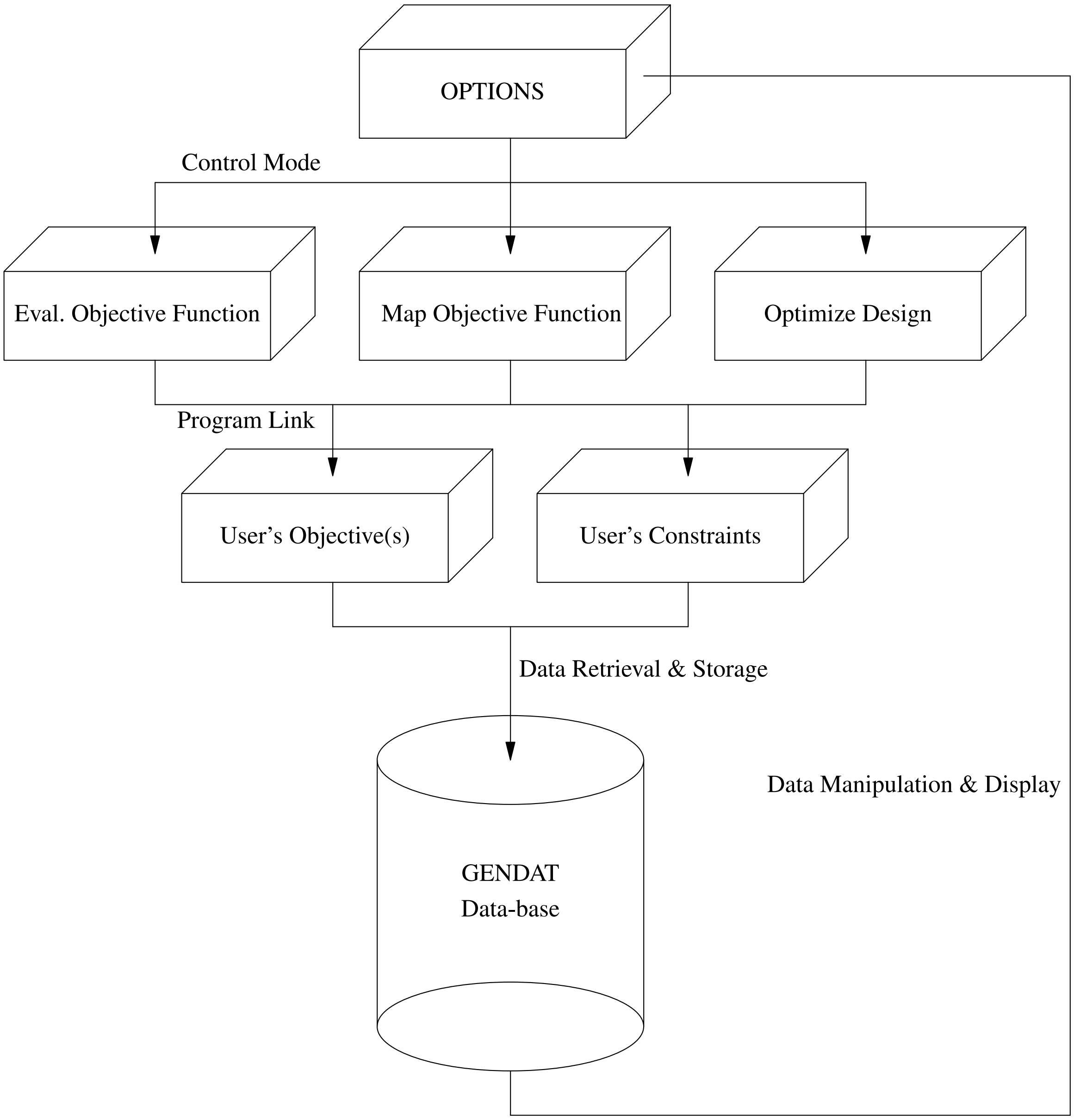



Figure 2.1 - **OPTIONS** Design System Structure.

OPTIONS MENU

| | |
|---|---|
| Control_Optimization | Evaluate_Obj_Funct |
| Plot_Obj_Funct | GAlg_Optimization |
| SAnn_Optimization | EProg_Optimization |
| EStrategy_Optimization | Show_Optimization |
| Vector_Order | |
| Misc_Optimizers | OPTIVAR_Optimizers |
| Schwefel_Optimizers | Graphics |
| Main_System | |

MISC. OPTIMIZERS MENU

| | |
|---|---|
| Control_Option | U1_Optimizer |
| U2_Optimizer | NAg_Optimizer |
| BClimb_Optimizer | DHClimb_Optimizer |
| PBIL_Optimizer | Num_Rcp_Optimizer |
| DoE_Optimizer | |
| Optimizer_Menu | |

OPTIVAR MENU

| | |
|---|---|
| Adrans_Optimizer | David_Optimizer |
| Fletch_Optimizer | Jo_Optimizer |
| PDS_Optimizer | Seek_Optimizer |
| Simplx_Optimizer | Approx_Optimizer |
| Random_Optimizer | Reduce_Optimizer |
| Control_Option | |
| Optimizer_Menu | |

SCHWEFEL MENU

| | |
|---|---|
| Fibonacci_Optimizer | Golden_Sect_Optimizer |
| Lagrange_Int_Optimizer | Hooke&Jeeves_Optimizer |
| Rosenbrock_Optimizer | DSCG_Optimizer |
| DSCP_Optimizer | Powell_Optimizer |
| DFPS_Optimizer | Simplex_Optimizer |
| Complex_Optimizer | 2M_ES_Optimizer |
| MM_ES_Optimizer | |
| Optimizer_Menu | |

GRAPHICS MENU

| | |
|---|---|
| Select_Device | Picture_Control |
| Captions | Key_Position |
| X-axis_Set | Y-axis_Set |
| Z-axis_Set | File_Options |
| Draw | Contour_Control |
| Surface_Control | |
| Transform_Data | Main_System |
| Define_Curve | |

DEFINE CURVE MENU

| | |
|---|---|
| X-data_Name | Y-data_Name |
| Z-data_Name | C-data_Name |
| Plot_Control | Range_of_Data |
| Curve_Type | Line_Type |
| Pen_Type | Symbol_Type |
| Caption | List_Details |
| Graph_Menu | |

TRANSFORM DATA MENU

| | |
|---|---|
| Scale_and_Shift | Power_Law |
| Log10(Abs(Value)) | Alog10(Value) |
| Col_Sum_Values | Row_Sum_Values |
| Copy_Sub_Array | Sum_Values |
| Subtract_Values | Multiply_Values |
| Divide_Values | User_Defined_Function |
| Main_System | |

Figure 2.2 - GENDAT Menu Structure of the **OPTIONS** System. Commands in the lower boxes allow movement between menus. All menus have access to the design data-base.

# CHAPTER 3

## Typical Design Approach

### 3.1. Introduction

As has already been mentioned the **OPTIONS** system can be used either manually, by evaluating and mapping the user selected objective function, or automatically, by deploying optimizers to search the design space for the design which best meets the specified requirements. Usually, interleaved sequences of interactions and optimizations are found most appropriate when developing a design. However, failures during this process can arise when very wide ranges are used for the quantities being varied as they may allow the system to reach unpredicted singularities. These are often caused by selecting inconsistent geometric parameters, which lead to impossible configurations. Another common problem is the selection of variables which cause the design to lie outside the range of validity of the user supplied code. For these reasons, monitoring and interaction are very important *during* this combined manual and automatic design process, so that suitable direction can then be given to the design process. Sometimes this monitoring throws up the need to refine or otherwise modify the design specific code because of some unanticipated combination of effects: this can be highly instructive and generate considerable insight into the problem in hand. As such DES's can be valuable design teaching aids.

### 3.2. Manual Approach

When the **OPTIONS** system is used in manual mode the user decides which quantities are to be fixed and which calculated. The fixed quantities are then input and the user defined modules applied in sequence, either by the function evaluation or function mapping options, to perform the desired calculations. The interelations between the various quantities used to define the design will obviously vary with the problem in hand. Nonetheless, the user must feel confident that, by manipulating the design data-base and running design calculations, reliable results are obtained in just the same way that the underlying problem code could have been used in a stand-alone fashion. Contour plots showing the trade off in performance achieved by systematically varying pairs of variables are very useful at this stage and may, in simple cases, be all that is required to complete the design exploration process. Often, more sophisticated searches are required and then the automated approach offered by the optimizers is appropriate.

### 3.3. Automated Approach

Having access to a number of powerful optimization routines, the obvious alternative might seem to be to select the automated mode allowing the chosen optimizer complete freedom to vary all the variables of interest. Even if this were desirable, it is not practicable for a number of reasons. First, a single loop through all of the design routines may well require a significant length of time. Therefore, runs of thousands of loops could require many hours to carry out and such numbers of loops are commonly required when optimizing many dimensional spaces over wide ranges. This may well not be justified until the user has some idea of what variables are most significant to the problem in hand. Secondly, it is always necessary to verify that the optimum found by an optimizer is the true global optimum, since some problems have local sub-optimal "peaks" and others constraints that can "stall" optimizers. Moreover, the true optimum may arise from an absurd set of parameters due to some unforeseen flaw in the design modules or use of them outside of their intended range. Finally, it has been found that not all of the optimization methods available within the package are suitable for every design process. Consequently, a more sophisticated strategy is required to get the best from the system described here:

(1) First, select an objective function, i.e., a parameter to be optimized, that is considered a reasonable measure of merit for the proposed design. This could contain terms representing numerous desirable characteristics in the final design but is more usually taken to be a single performance parameter, such as maximum power, overall efficiency or minimum initial or through life cost.

(2) Next, select as many constraints on a realistic and workable design as can be found (strength, stability, operating envelope, material choices, sizes, etc.).

(3) The constraints are then tested by supplying an initial design (normally an educated guess, a default version or the previous design) and using the manual, function evaluation, mode. If too many constraints have been selected or they have been drawn too tightly this will prove difficult (or perhaps impossible !) and this may lead to a revision of these quantities or their limits. It also provides a place for the designer to carry out an interactive design using the manual approach.

(4) To begin this manual design exploration, choose a set of design variables that are thought to have a significant influence on the objective function; these form the trial vector. The number of variables chosen determines the dimension of the function to be handled and since it is desirable, during preliminary work, to produce contour mappings of the function, the list of variables is next reduced to two. These are chosen to be the most dominant ones for the problem, usually by trial-and-error. This reduction allows the generation of a contour map of the objective function for the selected variables, subject to the given constraints. This map shows the type of function involved, indicating its behaviour as the variables change, revealing whether the parameters chosen do, in fact, control the objective function and also highlighting where constraints are active. It also allows subsequent optimization studies to be kept away from testing unpredictable combinations of variables that might otherwise cause difficulties.

(5) Next, optimize for just these two variables, so that the optimization process can be plotted on the contour map and different optimization strategies compared for the objective function under examination. During this process the system allows the user to monitor the optimization by displaying the status of the process at pre-defined numbers of loops while plotting the optimization path on the pre-prepared contour map. When the various optimizers finish, a status message is generated and if a true optimum has not been found, an explanation given. The reasons for failures vary; they can arise because the method chosen is not capable of further minimisation, or perhaps is unable to cope with an infeasible starting point, or simply that the amount of computer time specified by the user was too little to allow convergence. Often, the traditional methods will indicate that they have failed to converge yet will still have reached the vicinity of the optimum: failure messages should thus always be viewed alongside the final reported design. The availability of a contour map with the actual optimization path shown on it is crucial to understanding the behaviour of the optimizers during this initial trial stage.

(6) The previous two steps can then be repeated for other pairs of variables in the trial vector, thereby building up a picture of how the objective function behaves as a function of many variables.

(7) Finally, once the two dimensional processes have been fully studied, proceed to the many dimensional problem that arises when carrying out the full optimization, and for which maps cannot be produced.

At this point some confidence in the final answer may be expressed without resort to exhaustive testing. Moreover, effort will not have been wasted on searching variables which do not appear to affect the design greatly and unpredictable or invalid areas of the user supplied code will have been avoided. However, further manual manipulation of the design may be informative for the user, either in verifying the qualities of the suggested design or in re-defining the optimization problem, whereupon the above strategy can be re-entered. Often such explorations reveal that the optimum design is defined by one or more constraint boundaries and this may cause the designer to review these boundaries or even to remove them so as to enable speculation about profitable areas for development. Of course, some problems do not warrant this full strategy and then the optimization stages can be omitted, with design decisions being taken only on the basis of the relevant, manually produced, contour maps.

# CHAPTER 4

## An Example - Manual Approach

### 4.1. Introduction

Before giving a detailed, function by function, description of the facilities available within the **OPTIONS** system a brief example problem will be described and the system used to explore it. This example covers defining the problem, setting up the relevant design data-base and producing the user supplied code that describes it. It then illustrates typical manipulations of this design using both manual (interactive) and automatic (optimization) modes. The example problem is based on that given in a classic early paper on the subject of computer aided design exploration by Nowacki[14].

### 4.2. Problem Definition

Consider the design of tip loaded encastre cantilever beam: here the aim will be to design a minimum cost beam carrying a load, $F$, of 5 kN at a fixed span, $l$, of 1.5m. The beam is taken to be rectangular in section, breadth $b$, height $h$ and area $A$, and subject to the following design criteria:

(1) a maximum tip deflection, $\delta = \dfrac{Fl^3}{3EI_Y}$, of 5mm, where $I_Y$ is $\dfrac{bh^3}{12}$;

(2) a maximum allowable direct (bending) stress, $\sigma_B = \dfrac{6Fl}{bh^2}$, equal to the yield stress of the material, $\sigma_Y$;

(3) a maximum allowable shear stress, $\dfrac{3F}{2bh}$, equal to one half the yield stress of the material;

(4) a maximum height to breadth ratio, $h/b$, for the cross-section of 10 and

(5) the failure force for twist buckling, $F_{CRIT} = \dfrac{4}{l^2}\sqrt{\dfrac{GI_T EI_Z}{1-\nu^2}}$, to be greater than the tip force multiplied by a safety factor, $f$ of two, where $I_T$ is $\dfrac{b^3h + bh^3}{12}$ and $I_Z$ is $\dfrac{b^3h}{12}$.

The material used is to be selected from the following list:

(1) mild steel with a yield stress of 240 MPa, Young modulus, $E$, of 216.62GPa, bulk modulus, $G$, of 86.65 GPa, Poisson ratio, $\nu$, of 0.27, density of 7.85 tonne/m$^3$ and a unit price of 900 pounds/tonne;

(2) high tensile steel with a yield stress of 400 MPa, Young modulus of 210.0GPa, bulk modulus of 81.0 GPa, Poisson ratio of 0.3, density of 7.85 tonne/m$^3$ and a unit price of 1100 pounds/tonne and

(3) aluminium alloy with a yield stress of 395 MP, Young modulus of 72.0GPa, bulk modulus of 28.0 GPa, Poisson ratio of 0.33, density of 2.8 tonne/m$^3$ and a unit price of 1400 pounds/tonne.

Here, the trial vector describing the design consists of the cross-section breadth and height plus the material choice. Notice that it is not clear from the above specification which of the design limits will control the design, although clearly at least one will, if the beam is to have a non-zero cross-sectional area. Moreover, because the design limits are not obvious neither is the correct material selection. This problem is also complicated by the fact that, although the depth and breadth of the section may be varied continuously, the material properties cannot (of course, in many instances stock beam sizes might have to be used and this would impose further difficulties).

Initially, this problem will be studied with the fixed material choice of mild steel. In subsequent chapters the third, discontinuous variable will be introduced, leading to a significantly more complex problem (albeit one where the correct solution is still known).

### 4.3. Data-base Design

To begin the analysis of this example using **OPTIONS** the user must first set out all the variables of the problem, the mathematical relationships that hold between them and the order they must be evaluated in. This must include the limits on constraints and any range restrictions on the design variables (these latter restrictions are not necessary but are highly desirable when deploying the optimizers or plotting the objective function, since they define the ranges of variables that must be explored; if in doubt it is better to set very wide ranges than none at all).

Having done this the user must then split the variables into two groups: first, there are those that the user will wish to study and/or modify and secondly, those which are completely internal to the design calculation and which need not be monitored explicitly. This latter group of variables does not have to figure in the design data-base so they can be completely internal to any calculations carried out in the design specific code. For a small problem all variables may be placed in the first category for simplicity and also to allow complete control. For a major design with many hundreds of variables this is not always feasible although the **OPTIONS** software will allow thousands of separate variables if desired. Table 4.1 list the variables in this initial problem, together with the names they will be identified by within the **OPTIONS** data-base, their units and meanings. Note that all of the variables that are limits *must* be given names that relate them to the variables to which they apply. This is achieved by using the same name as the relevant variable prefixed by "L_", "U_", "E_" or "D_" to indicate lower bounds, upper bounds, equality constraints or discrete variables, respectively (discrete variables being those which can only take one of the fixed set of values in the "D_" list). Clearly, design variables may not be the subject of equality constraints. Notice that these names need not be used within the user supplied problem specific code, but that they should be similar if confusion is not to arise.

| Variable | Name | Value | Units | Type | Meaning |
|---|---|---|---|---|---|
| $A$ | AREA | | $mm^2$ | Obj_Funct | Cross-sectional area of the beam. |
| $b$ | BREADTH | 30.0 | mm | Opt_Var | Breadth of the beam. |
| $h$ | HEIGHT | 7.0 | cm | Opt_Var | Height of the beam. |
| $l$ | LENGTH | 1500.0 | mm | Parameter | Length of the beam. |
| $F$ | FORCE | 5000.0 | N | Parameter | Force on the tip of the beam. |
| $E$ | EE | 216620. | MPa | Parameter | Young's modulus. |
| $G$ | GG | 86650. | MPa | Parameter | Modulus of rigidity. |
| $\nu$ | NU | 0.27 | | Parameter | Poisson's ratio. |
| $\sigma_Y$ | SIGMA-Y | 200.0 | MPa | Parameter | Yield stress. |
| $f$ | FACTOR | 2.0 | | Parameter | Safety factor on the critical force for twist buckling. |
| $\sigma_B$ | SIGMA-B | | MPa | Constraint | Max. bending stress. |
| $\tau$ | TAU | | MPa | Constraint | Max. shear stress. |
| $\delta$ | DEFLN | | mm | Constraint | Tip deflection. |
| $h/b$ | H-ON-B | | | Constraint | Height to breadth ratio. |
| $F_{CRIT}$ | F-CRIT | | N | Constraint | Critical force for twist buckling. |
| | L_BREADTH | 5.0 | mm | Limit | Min. permissible beam breadth. |
| | U_BREADTH | 50.0 | mm | Limit | Max. permissible beam breadth. |
| | L_HEIGHT | 2.0 | cm | Limit | Min. permissible beam height. |
| | U_HEIGHT | 25.0 | cm | Limit | Max. permissible beam height. |
| | U_SIGMA-B | | MPa | Limit | Max. permissible bending stress. |
| | U_TAU | | MPa | Limit | Max. permissible shear stress. |
| | U_DEFLN | 5.0 | mm | Limit | Max. permissible tip deflection. |
| | U_H-ON-B | 10.0 | | Limit | Max. permissible height to breadth ratio. |
| | L_F-CRIT | 5000.0 | N | Limit | Min. permissible critical force for twist buckling. |

Table 4.1 - Initial Data-base Design.

Next, the variables are classified as belonging to one of six types:

(i) design (or optimization) variables - type Opt_Var
which are those that the user intends to control the design with;

(ii) objective function - type Obj_Funct
which forms the goal of the design optimization process;

(iii) constraints - type Constraint
which are variables that are restricted in some way by the design specification;

(iv) limits - type Limit
which are variables that define the limits on the design or constraint variables;

(v) parameters - type Parameter
which are quantities that are used when performing the calculations, but which will be constant throughout the calculation sequences (in most problems where no new, problem specific, types are defined this type will cover the bulk of variables in the data-base) and

(vi) dependent variables - type Dependent
which are those that are calculated afresh for each change to the design, and which do not fall into any of the other categories.

The initial choices for this example problem are also detailed in Table 4.1. Note that, although an initial typing must be chosen, this can be easily modified during operation of the package. This feature allows the choice and number of design variables or of the objective function, etc., to be readily altered when using the compiled program (clearly, this fact must be born in mind when writing the problem specific code if numerous recompilations and relinks are to be avoided). Note that, as to begin with the material is fixed to be mild steel, the objective function is merely set as the cross-sectional area, since this is directly related to cost, given a fixed material price.

### 4.4. Problem Specific Code

Having decided on the variables, named them, fixed whether they are scalars or arrays (and if so their dimension), typed them and specified their meanings and what units they are given in, this information can be used to set up the design data-base for the problem in hand. However, to do this the **OPTIONS** package must be run and this requires at least two user supplied routines to be built and linked to the package (OPTPRE and OPTPST need not be specified as null versions are included in the **OPTIONS** library and these form the default; it is also possible to supply a combined version of the two mandatory routines if this suits the problem being tackled although this is always somewhat less computationally efficient).

The first of these two mandatory pieces of code, OPTFUN, deals with evaluating the chosen objective function: in the first version studied here it takes a specified cross-section breadth and height and then calculates the beam's cross-sectional area. The second, OPTCON, deals with the constraints and takes the previous inputs along with the beam's length and the tip load together with the material properties and the buckling safety factor. It then runs the calculations on the design limits, allowing the package to see if any of these are not being met. These first two code segments are given as Appendix E and these routines are also supplied with the standard installation as "beam1fun.f" and "beam1con.f" ("beam1fnc.f" gives the equivalent combined version). As has already been noted, this subdivision allows some design explorations to be run more quickly, since they may not need to evaluate the objective function if the choice of design variables violates some or all of the design requirements; this can dramatically reduce run time when such calculations are time consuming. (If, however, it is tedious to separate the user supplied calculations into two completely independent routines a single joint routine called OPTFNC can be used instead. This routine is then called when either the objective function or constraints must be analysed - an internal cache prevents unnecessary duplicate subroutine calls.)

The two user supplied routines shown in the appendix follow a standard pattern, albeit one that may be unfamiliar at first sight. Note first that they only take a single argument, the logical flag ERROR, which is checked after the routine is used and is only set if the routine encounters an error condition (i.e., if it could not correctly carry out the calculations normally required, perhaps because the supplied data was out of range or inadequate data was available in the data-base). Instead of passing incoming variables in the argument list they are handled by calls to the function GETVV/GETDEF (or here sometimes, their double precision variants, GETVVD/GETDFD), which takes the name of the desired variable as a character string, together with its array subscripts (which are set to one for scalars) and in the case of GETDEF/GETDFD, default values, finally followed by an error or default flag. These routines extract the desired data item from the data-base, checking that the variable exists and has a valid numeric value as they do so. If the variable does not exist or has no value set GETVV/GETVVD return with the error flag set while GETDEF/GETDFD return the given default value with the default flag set. This method is robust and leads to few potential coding errors. It also allows the data-base to keep track on the usage of variables. Notice that GETDEF/GETDFD should only be used when it is acceptable for a fixed default value to be used in the design. Therefore they are sensible for values such as the material properties $E$, $G$ and $\nu$ but not for the breadth and height of the section that will be changed.

A similar process is used to return computed values to the data-base: these are sent back via calls to the subroutines SETVV/SETVVD or SETDEF/SETDFD which are similar in format to GETVV, etc. Again, these routines allow various checks to be made and the data to be safely stored. In this first example only SETVV/SETVVD are used as all the results being set are required for the search to take place. SETDEF/SETDFD are used to send data to the data-base that might be useful but might not. Then if the user has not bothered to define the relevant data-base variables needless errors do not arise. (The structure of OPTFNC is entirely similar to that of OPTFUN and OPTCON.)

Notice that should either the GETVV/GETVVD or SETVV/SETVVD routines fail to work correctly an error message is generated and as their flags are linked to the main error flags of the calling routines this allows the optimizers, function mappers, etc., to take appropriate action. Finally, note that neither routine specifies which variables are to form the design vector, objective function or constraints. These are set interactively within the **OPTIONS** menus and can change from run to run without modifying the coding of these two routines and thus without relinking the main package, provided that OPTFUN does evaluate the desired objective and OPTCON the relevant constraints (of course, such a distinction is not relevant if using OPTFNC).

It is also important to remember that the **OPTIONS** package uses the fortran logical unit numbers 10 and 11 for a number of purposes and therefore these LUNs should be avoided in user written code.

Having written these two routines they are compiled and then linked with the main **OPTIONS** library in the conventional way (under UNIX using something like "f77 -o options main.o opt.f beam1fun.f beam1con.f -lgendat" followed by any other graphics or maths libraries depending on the specific details of the installation - a site specific "make file" is normally supplied containing such information).

### 4.5. Building the Data-base

Having built the running program the user must next set up a data-base structure to match the problem in hand. This may be built from scratch, from the skeleton provided with the program ("opdata.d") or be a modification of a previously defined one; here the structure is built from scratch (and includes the system variable "WORK" which *must* be present if objective function plotting is to be carried out). (It is also possible to use **OPTIONS** via direct subroutine calls which obviate the need for this stage, see Chapter 8.) This aspect of setting up the problem often takes a little while and may, at first sight, seem rather tiresome. This is because full details of the problem must be entered for the data-base to be workable. However, this is quite deliberate and the data-base then forms a self-documenting record of what has been done: this may be vital for later study, particularly if a new user is attempting to pick up a previous design and modify it.

To carry out this process the user adds the required data types followed by a sequence of new variables using the functions intrinsic to GENDAT (note that the six types defined here *must* be adopted for the package to work, although additional types may be added to help the clarity of the data structure). As each new variable is added all the information previously decided upon is entered as requested by the package. Variables need not be added in any specific order as the package allows the display order to be re-arranged at any time by assigning record numbers to each variable. Moreover, all the characteristics of each of the variables may be modified at any stage. Appendix F contains a suitably commented and slightly foreshortened log of the interactions required for this example. Note that these sequences of commands could have been prepared using the system editor and then execute as a macro file, a feature that is used in latter examples. Also, if the XView interface is not being used or instructions are being typed in manually, the commands do not need typing in the full form shown in the log of this example: sufficient characters to distinguish between menu options being all that is needed (entering a "?" will display these if the XView interface is not being used).

The final stage of data-base preparation requires the user to input values for all the parameters and limits that will not be calculated by the routines, plus an initial set of design variables. Clearly, the more workable and optimal the initial design values the more rapidly an optimum design may be produced; often, however, these values are just guesses. The input of these variables is also shown in Appendix F which finishes with the instructions to store the initial data-base as a file called "beam1.d" and then to produce printed hard-copy as a file called "beam1.txt". The print file generated is included as Figure 4.1. The self documenting nature of the data-base is clearly seen in this figure. Notice that variables that have not been given values have blanks in the "Value" column and that the "Element" column is empty because all the variables thus far defined are scalars. The "RcNo" column just counts through the variables since variables are always printed and displayed in record number order. The "SCtr" and "GCtr" columns indicate the set and get counters, showing how many times the data has been set to the data-base or got from it. The "Data usage" section allows a free format comment to be added to the data-base and the "Data name" item records the last data-file name used by GENDAT and thus the name assumed for the data-base. Lastly, the variable WORK, which is a system variable, is included to allow for work information to be stored during plotting.

### 4.6. Checking the Data-base

Having produced and stored the initial file the "Show_Optimization" option is invoked. This option checks that the problem is consistently set up: it looks to see that there is only one objective, that there are some design variables and examines the constraints and limits. It warns if any essential variables are defined but do not have numeric values set or if constraints do not have matching limits, etc. At this stage the user may well need to correct any errors in the data-base. Note, that this option does not cause the user supplied code to be executed and so does not check for coding errors, etc. Appendix F also shows this interaction.

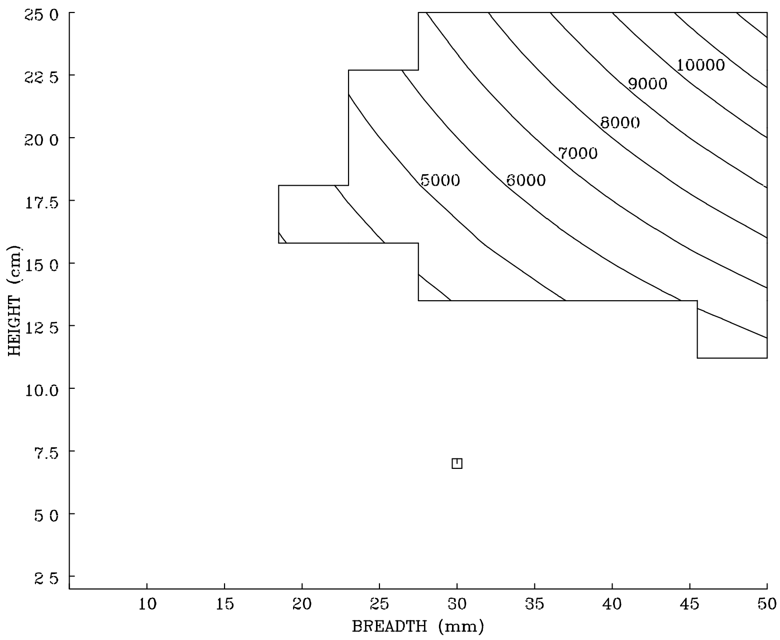
Once the newly constructed data-base passes the checks carried out by the "Show_Optimization" command the user may see if the problem specific routines are working correctly. These may be invoked by using the "Evaluate_Obj_Funct" command, which simply calls the routines OPTPRE, OPTFUN, OPTCON and OPTPST in order. The package makes sure that the objective function is calculated by OPTFUN and all the constraints are set by OPTCON (but note that a variable whose type does not mark it out as a constraint or as the objective function is not checked until its type is changed to one of these). If these routines are correctly coded and access the data-base without error the user may again use the "Show_Optimization" function to see the results of the calculation. Alternatively, the GENDAT "Type_Data" command may be used to scroll through the data-base to see what has happened. Clearly, a number of problems could reveal themselves at this stage. The two most common are: first, the user's routines attempts to access a data-base variable that does not exist (usually because of a name typo.) or one that has no value set (because no value has been input, or calculated and set, before being subsequently required) and secondly, the routines do not calculate all the quantities required (so that the data-base values are incorrectly set). Appendix F also shows this process. Here, the objective function, the cross-sectional area, takes the value of 2100mm$^2$ and the design is seen to fail to meet two of the design criteria: clearly, some improvement will be required.

### 4.7. Mapping the Objective Function

Once the user has a data-base plus working user supplied routines the previously outlined design strategy may be commenced. This begins with the preparation of contour maps, iso-metric projections or cross-sections of the objective function. Initially, the beam problem being studied here is simply a function of two, continuous variables (cross-section breadth and height) and so a simple two-dimensional contour map will completely illustrate the problem. A first attempt to map the function may be readily generated using the "Plot_Obj_Funct" option with its default, auto-scaling sub-options. This evaluates the function at 11 by 11 equi-spaced points and produces the result included as Figure 4.2. The "□" on this figure indicates the current (initial) design with a breadth of 30mm and a height of 7cm and this is clearly not feasible, as already noted. The boundaries to the feasible region on this plot are stepped in shape because the package has plotted the surface with few points and this is the best that may be inferred. Using more points will not remove this characteristic but will reduce the size of the steps. This is illustrated in Figure 4.3 which has been produced using the manual range control facilities using 41 by 41 points (see again, Appendix F). This more refined plot shows that an optimal solution will have a breadth of between 15 and 20 mm and a height of between 15 and 18 cm (note that, because only feasible points are used to produce the contours, the actual feasible region will *always* be bigger than that contoured, here this will lead to lower optimal values for both variables).

At this point the design process could be stopped and a good solution produced without recourse to further studies. Alternatively, a further, manual, exploration could be carried out over the ranges noted in the previous paragraph. Optimizers will be used in the next chapter to automate this design search.

Figure 4.1 - Print file showing initial data structure.




CONTOUR MAP OF OBJECTIVE FUNCTION – AREA (mm↑2)

Figure 4.2 - Initial map of objective function (area versus breadth and height).

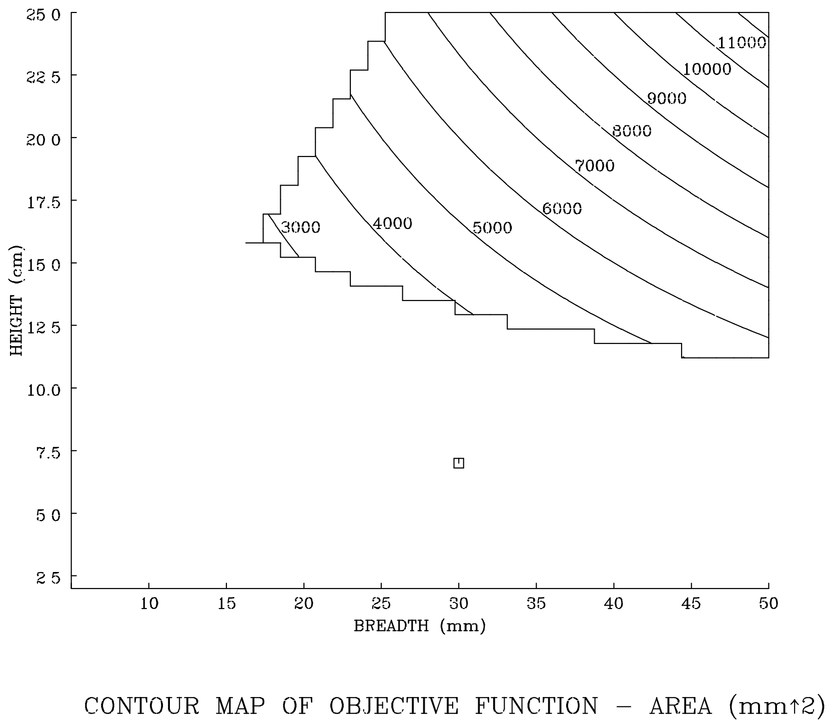



Figure 4.3 - Refined map of objective function (area versus breadth and height).

# CHAPTER 5

## An Example - Automated Approach: Part I

### 5.1. Introduction

Continuing from the previous chapter, and to demonstrate the various methods available within the automated approach, the initial, two variable, beam design problem already mapped is next explored using several of the optimizers available within the **OPTIONS** package. Again the goal is set as minimum cross-sectional area within the previously specified constraints (i.e., stability, strength, tip deflection, etc.). Since a map of the objective function has already been produced the optimization paths are traced out on this map as they proceed.

Of course, it is to be expected that the predictable changes will be made to improve this design. However, this is a consequence of asking a question to which the answer is already established: the real strength of optimizers lies in their ability to deal with many competing aspects at one time, whether or not the user is able to predict the likely consequences of attempting to satisfy such requirements, i.e., a known task has been specified to illustrate the process and to show that the *optimizers* can achieve the desired result without *a priori* knowledge. A record of the terminal interaction for these runs is given as Appendix G.

### 5.2. Choice of Optimizer

Here, as has already been noted, the initial input data for the design do not allow all the criteria used as constraints to be met (i.e., it is *not* a feasible design). This restricts the choice of optimizer since some do not work when given an initially infeasible design. However, limits have also already been chosen for the design variables and the optimizers will therefore seek to remain within these limits, reducing the scope of the task somewhat (some of the methods within the OPTIVAR library treat these range constraints as advisory by default and may, therefore, explore beyond their limits; this may be over-ridden if desired). To give a representative cross-section of the methods available the following techniques have been applied:

(1) the method of successive linear approximation - APPROX;

(2) the genetic algorithm - GA, with a one pass, external penalty function - OPTIM1 and

(3) the Hooke and Jeeves direct search - SEEK, with the Fiacco-McCormick combined external and internal penalty function - OPTIM2.

All of the optimization methods available in the **OPTIONS** package have a number of control parameters to set step sizes, number of loops, tolerances, etc., and these may be selected as required. Here defaults are used (except that SEEK is used with 1,000 steps) and this causes various warnings to this effect to be generated.

### 5.3. Optimization with Two Variables

The optimization problem being tackled here is summarized in Table 5.1. Given the previous contour map of this problem the optimum is obvious and the purpose here is to establish that the optimizers can, in fact, reach this sub-goal. The results achieved are summarized in Table 5.2.

APPROX
: This method gives a successful optimization, the minimum being reached very rapidly, in 28 loops. The optimization path is shown in Figure 5.1 and it can be seen that the expected minimum is reached.

GA
: Using the genetic algorithm and a one pass penalty function (OPTIM1) a less precise solution is found and this represents one of the fundamental limitations of the method. Nonetheless, the method gives a reasonably passable result. Here, only the best of generation points are shown on the optimization path, along with all points from the last generation, see Figure 5.2.

| Optimization | with 2 variables |
|---|---|
| Objective Function | cross-sectional area |
| Trial Vector and Limits | 5 < breadth (mm) < 50<br>2 < height (cm) < 25 |
| Constraint Vector and Limits | direct stress < yield stress<br>shear stress < yield stress/2<br>tip deflection < 5mm<br>height to depth ratio < 10<br>tip force × safety factor < buckling force |
| Fixed Parameters | tip load = 5 kN<br>length = 1.5m<br>material type = 1 (mild steel) |

Table 5.1 - Optimization Problem Summary.

| Method | | APPROX | GA | SEEK |
|---|---|---|---|---|
| Penalty Function | | n/a | OPTIM1 | OPTIM2 |
| Success/Failure | | Success | Success | Success |
| Number of Loops | | 32 | 500 | 773 |
| Objective Function | area (mm$^2$) | 2496.42 | 2689.64 | 2496.87 |
| Trial Vector | breadth (mm) | 15.80 | 16.42 | 15.80 |
| | height (cm) | 15.80 | 16.38 | 15.80 |
| Constraint Vector | direct stress (MPa) | 114.1 | 102.1 | 114.1 |
| | shear stress (MPa) | 3.004 | 2.788 | 3.004 |
| | tip deflection (mm) | 5.000 | 4.318 | 5.001 |
| | height to breadth ratio | 10.00 | 9.975 | 9.999 |
| | buckling force (kN) | 132.0 | 153.3 | 132.0 |

Table 5.2 - Optimization with Two Variables.

SEEK Used with the Fiacco-McCormick combined external and internal penalty function (OPTIM2) this method also gives a successful optimization. The process ends after 773 loops, requiring so many steps because of the repeated sub-optimizations used. The results are similar to those given by APPROX. The general path of the optimization process is shown in Figure 5.3.

The only problem encountered with these two dimensional optimizations concerns the inability of the GA to reach the exact optimum. Clearly, APPROX is a very rapid method and it transpires that this method should always be used first when dealing with two-dimensional problems as it either succeeds or fails very quickly. It is also highly useful in fine tuning the results produced by other methods.

This concludes the study of the two variable version of the example problem: next attention is turned to the more complex case with three variables.

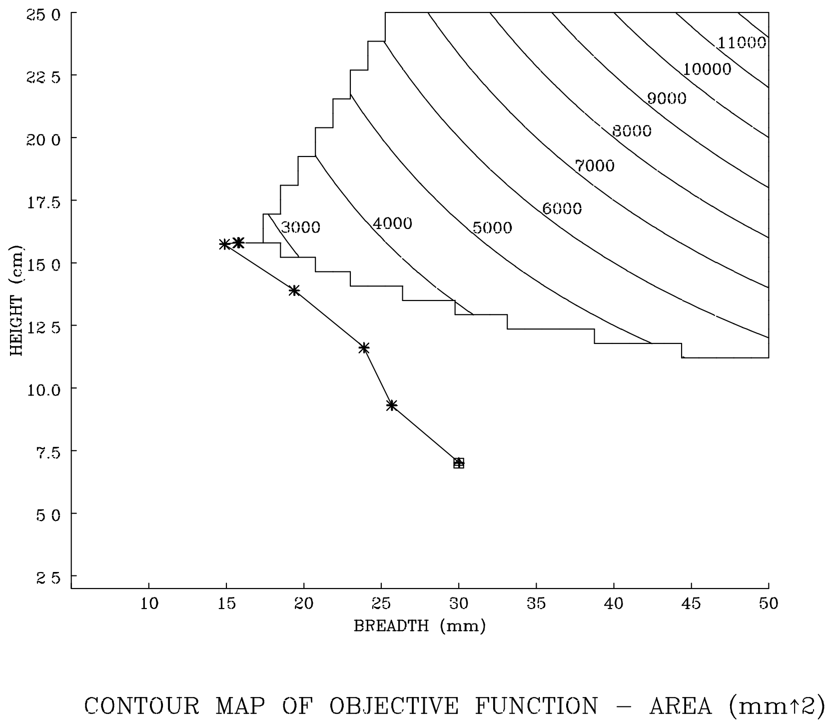



Figure 5.1 - Optimization path with two variables (APPROX).

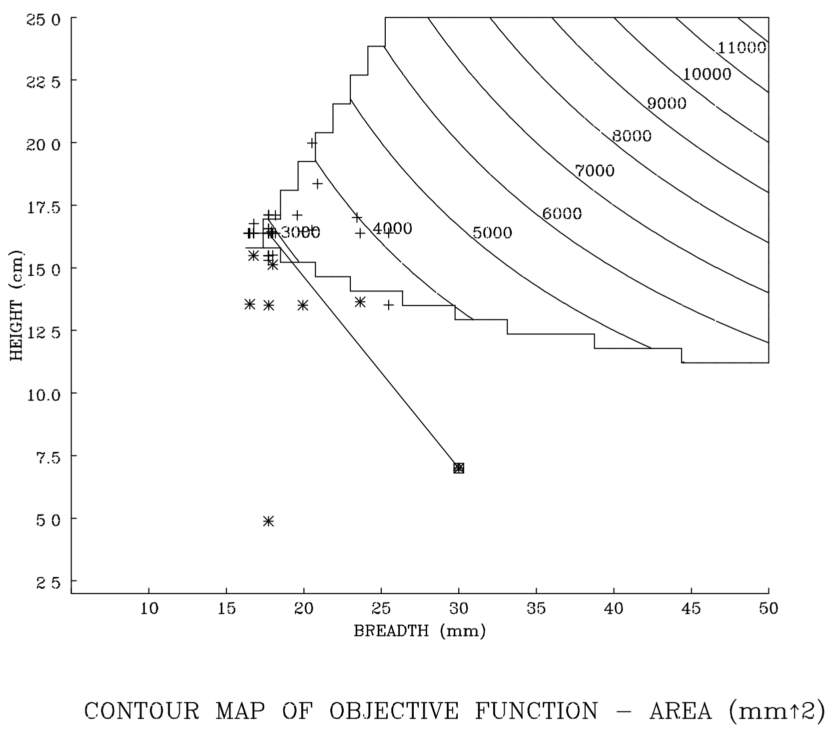



Figure 5.2 - Optimization path with two variables (GA & OPTIM1).

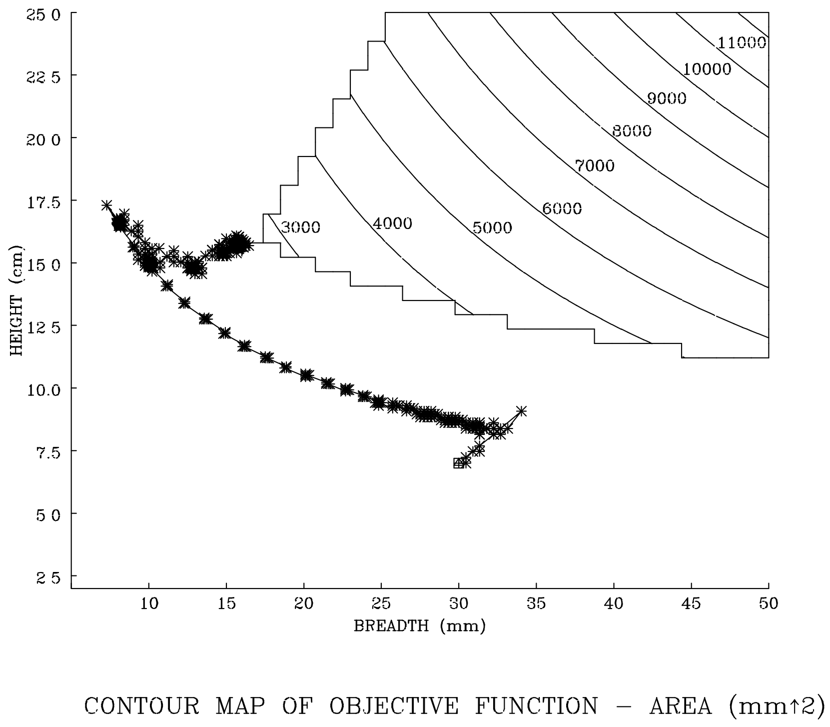



Figure 5.3 - Optimization path with two variables (SEEK & OPTIM2).

# CHAPTER 6

## An Example - Automated Approach: Part II

### 6.1. Introduction

Having demonstrated the use of optimizers for the two-dimensional version of the example problem attention is next focussed on the three-dimensional case where a material selection must also be made. Now the goal is set as minimum cost and both the data-base and problem specific coding must be revised to deal with the additional complexities involved. The same three optimization methods will be used but here the three-dimensional optimization path cannot be easily illustrated. However, since only three materials are involved, the entire objective function can be mapped with three plots. Copies of the macro command files used for these runs are given in Appendix H. The use of macro files, which are sequences of commands prepared with the system editor, is very common when some experience has been gained with the system as it allows rapid control and operation of the package; it is perhaps best avoided by novice users.

### 6.2. Modifications to Design Data-base

To allow for three material types and also to produce costings a number of new variables must be included in the data-base. Moreover, several of the existing and new variables must be specified as arrays rather than scalars to allow for these changes. Here, the material type to be used is controlled by the variable "MAT_TYPE" which takes the integer values one to three to indicate the three materials previously described. Note that the data-base only deals with real quantities and so special steps must be taken to deal with this aspect when coding the user supplied routines. Table 6.1 details the revised data-base structure and Table 6.2 summarizes the optimization problem alongside that already studied.

### 6.3. Modifications to Problem Specific Code

Having decided on the additional variables to be used and assigned names to them, the two routines OPTFUN and OPTCON are next modified to reflect the desired changes. The modified versions are included as Appendix I. Note that in both routines the material type number is accessed as a floating point number, converted to the nearest integer (to avoid any rounding problems) and then restricted to lie between one and three. These steps safeguard against the quantity taking unexpected values which would then result in errors when it was used in subsequent statements as an array subscript. They also allow this variable to be treated by the optimizers as continuous if need be.

Also declared in the data-base is the array "D_MAT_TYPE" which holds the three allowable values of the material type. This array is declared to be of type "Limit" so that the optimizers try and and hold the material type variable to one of these discrete values. Not all optimizers are able to enforce this restriction but it useful for those that can to be informed of the list of possibilities. Those routines that can work in this way are primarily the stochastic search methods and this capability is highlighted in the optimizer descriptions in Appendix D where it is possible.

Note also that now cost is used as the objective function but that the cross sectional area is still calculated but is now stored in the data-base via a call to SETDFD. This means that the area values are only stored if the data-base variable is defined - this again reflects the fact that this variable is no longer an essential piece of information in the problem. Using SETDFD (or SETDEF) in this way is a good habit since it minimizes the size of data-base actually required while still allowing data to be handled this way if desired without re-editing and re-compilation of the code. Apart from this and the use of subscripts other than one when retrieving material properties, the routines are not changed from their previous form.

| Variable | Name | Value | Units | Type | Meaning |
|---|---|---|---|---|---|
| $A$ | AREA | | $mm^2$ | Dependent | Cross-sectional area of the beam. |
| | COST | | pounds | Obj_Funct | Material cost. |
| $b$ | BREADTH | 30.0 | mm | Opt_Var | Breadth of the beam. |
| $h$ | HEIGHT | 7.0 | cm | Opt_Var | Height of the beam. |
| $l$ | LENGTH | 1500.0 | mm | Parameter | Length of the beam. |
| $F$ | FORCE | 5000.0 | N | Parameter | Force on the tip of the beam. |
| | MAT_TYPE | 1.0 | | Opt_Var | Material type (1=mild steel, 2=HT steel, 3=al. alloy). |
| $E$ | EE(1) | 216620. | MPa | Parameter | Young's modulus. |
| | EE(2) | 210000. | | | |
| | EE(3) | 72000. | | | |
| $G$ | GG(1) | 86650. | MPa | Parameter | Modulus of rigidity. |
| | GG(2) | 81000. | | | |
| | GG(3) | 28000. | | | |
| $\nu$ | NU(1) | 0.27 | | Parameter | Poisson's ratio. |
| | NU(2) | 0.30 | | | |
| | NU(3) | 0.33 | | | |
| $\rho$ | RHO(1) | 7850.0 | kg/m^3 | Parameter | Density. |
| | RHO(2) | 7850.0 | | | |
| | RHO(3) | 2800.0 | | | |
| | PRICE(1) | 900.0 | pnds/t | Parameter | Price per tonne. |
| | PRICE(2) | 1100.0 | | | |
| | PRICE(3) | 1400.0 | | | |
| $\sigma_Y$ | SIGMA-Y(1) | 200.0 | MPa | Parameter | Yield stress. |
| | SIGMA-Y(2) | 400.0 | | | |
| | SIGMA-Y(3) | 395.0 | | | |
| $f$ | FACTOR | 2.0 | | Parameter | Safety factor on the critical force for twist buckling. |
| $\sigma_B$ | SIGMA-B | | MPa | Constraint | Max. bending stress. |
| $\tau$ | TAU | | MPa | Constraint | Max. shear stress. |
| $\delta$ | DEFLN | | mm | Constraint | Tip deflection. |
| $h/b$ | H-ON-B | | | Constraint | Height to breadth ratio. |
| $F_{CRIT}$ | F-CRIT | | N | Constraint | Critical force for twist buckling. |
| | L_BREADTH | 5.0 | mm | Limit | Min. permissible beam breadth. |
| | U_BREADTH | 50.0 | mm | Limit | Max. permissible beam breadth. |
| | L_HEIGHT | 2.0 | cm | Limit | Min. permissible beam height. |
| | U_HEIGHT | 25.0 | cm | Limit | Max. permissible beam height. |
| | U_SIGMA-B | | MPa | Limit | Max. permissible bending stress. |
| | U_TAU | | MPa | Limit | Max. permissible shear stress. |
| | U_DEFLN | 5.0 | mm | Limit | Max. permissible tip deflection. |
| | U_H-ON-B | 10.0 | | Limit | Max. permissible height to breadth ratio. |
| | L_F-CRIT | 5000.0 | N | Limit | Min. permissible critical force for twist buckling. |
| | L_MAT_TYPE | 1.0 | | Limit | Min. permissible material type. |
| | U_MAT_TYPE | 3.0 | | Limit | Max. permissible material type. |
| | D_MAT_TYPE(1) | 1.0 | | Limit | Defined material types. |
| | D_MAT_TYPE(2) | 2.0 | | | |
| | D_MAT_TYPE(3) | 3.0 | | | |

Table 6.1 - Revised Data-base Design.

| Optimization | with 2 variables | with 3 variables |
|---|---|---|
| Objective Function | cross-sectional area | cost |
| Trial Vector and Limits | 5 < breadth (mm) < 50<br>2 < height (cm) < 25 | 2 < height (cm) < 25<br>1 < material type < 3 |
| Constraint Vector and Limits | direct stress < yield stress<br>shear stress < yield stress/2<br>tip deflection < 5mm<br>height to depth ratio < 10<br>tip force × safety factor < buckling force | |
| Fixed Parameters | tip load = 5 kN<br>length = 1.5m | |

Table 6.2 - Optimization Conditions (Summary).

### 6.4. Revised Mappings

Having compiled the new version of the program the required modifications are then made to the data-base, see Figure 6.1 and the first macro file in Appendix H. To completely illustrate the function it is then mapped for the three different values of the variable "MAT_TYPE". This is readily accomplished using the "Plot_Obj_Funct" command which simply requires the user to select which variables to plot when more than two are available. The resulting three maps are given as figures 6.2-4. It is clear from these figures that both mild steel and aluminium alloy are contenders for the best design in this problem. In fact it turns out that the alloy solution is just the best and here this has been artificially contrived to make the problem more difficult.

### 6.5. Optimization with Three Variables

Given the contour maps of this problem the optimum is again fairly obvious but now the optimizers have a more difficult goal in reaching their objectives. This is exacerbated by the fact that small, fractional changes to the variable "MAT_TYPE" have no effect until they reach 1.5 or 2.5. The results achieved are summarized in Table 6.2 which are the result of using the second macro file in Appendix H.

| Method | | APPROX | GA | SEEK | GA & APPROX |
|---|---|---|---|---|---|
| Penalty Function | | n/a | OPTIM1 | OPTIM2 | OPTIM1 |
| Success/Failure | | Success | Success | Success | Success |
| Number of Loops | | 40 | 500 | 1132 | 500 & 10 |
| Objective Function | cost (pounds) | 26.4558 | 25.9579 | 26.4552 | 25.4612 |
| Trial Vector | breadth (mm) | 15.80 | 21.19 | 15.80 | 20.81 |
| | height (cm) | 15.80 | 20.83 | 15.80 | 20.81 |
| | material type | 1 | 3 | 1.4986 (≡ 1) | 3 |
| Constraint Vector | direct stress (MPa) | 114.1 | 48.93 | 114.1 | 49.94 |
| | shear stress (MPa) | 3.004 | 1.699 | 3.004 | 1.732 |
| | tip deflection (mm) | 5.000 | 4.893 | 5.000 | 5.000 |
| | height to breadth ratio | 10.00 | 9.832 | 10.00 | 10.00 |
| | buckling force (kN) | 132.0 | 138.0 | 132.0 | 132.8 |

Table 6.2 - Optimization with Three Variables.

APPROX

This method is unable to cope with the highly non-linear nature of the material type. Consequently, it keeps to mild steel and converges to the previously found optimum, this time taking slightly longer because of the additional variable being tested.

GA Using the genetic algorithm and the one pass external penalty function (OPTIM1) the correct material choice is made but again an imprecise solution is found.

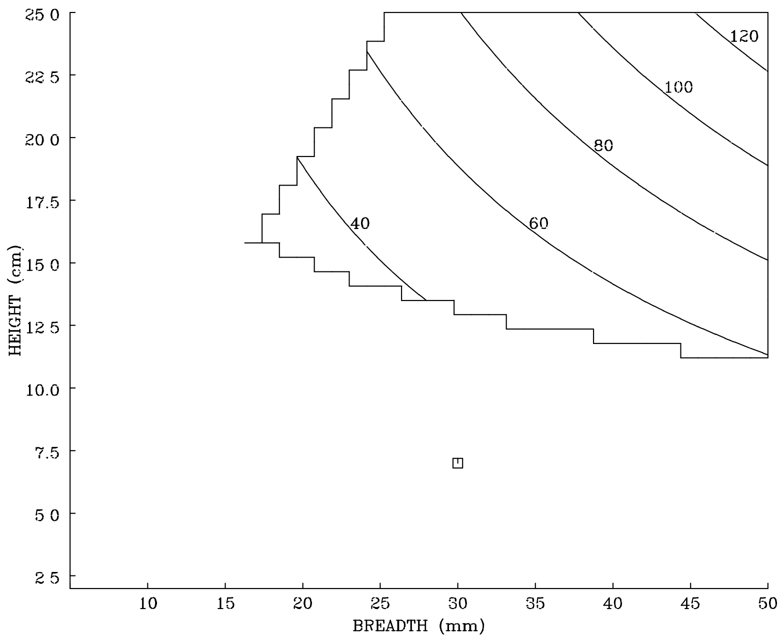
SEEK Using the Fiacco-McCormick combined external and internal penalty function (OPTIM2) this method also fails to select the correct material, again because it works with small changes which are found to have no effect. The process ends after 1,132 loops, again requiring more steps because of the repeated sub-optimizations used (note only 1,000 steps were specified: SEEK is not rigidly controlled by the specified number of steps). The results are similar to those given by APPROX.

GA followed by APPROX

If the solution reached by the GA is followed by APPROX it gets to the correct answer very rapidly. The combination of methods is seen to be very powerful, with the stochastic method giving the correct region and the classical slope driven one then searching out the true optimum.

This, highly non-linear, example reveals the major problem of using slope driven optimizers. Unless the local slope indicates the correct direction in which to move they become hopelessly confused. The way this non-linear function appears to such methods is well illustrated by mapping the objective function at the true optimum, keeping the height of the beam fixed at its optimum value, see figures 6.5-9. Figure 6.10 shows the value of the objective for the previously optimal dimensions when using just mild steel and this illustrates that this is not the true, three-dimensional optimum. The final section of the second macro file in Appendix H generates these figures. Note that figure 6.5 is misleading because of the restricted number of elements used to form the plot, see figure 6.6. Figures 6.7 and 8 further illustrate the function where here, the "MAT_TYPE" variable has no longer been restricted to just the three values in the "D_MAT_TYPE" array, allowing a more detailed plot to be produced.

Figure 6.1 - Print file showing revised data structure.




CONTOUR MAP OF OBJECTIVE FUNCTION – COST (pounds)

Figure 6.2 - Refined map of objective function for material type 1, mild steel (cost versus breadth and height).

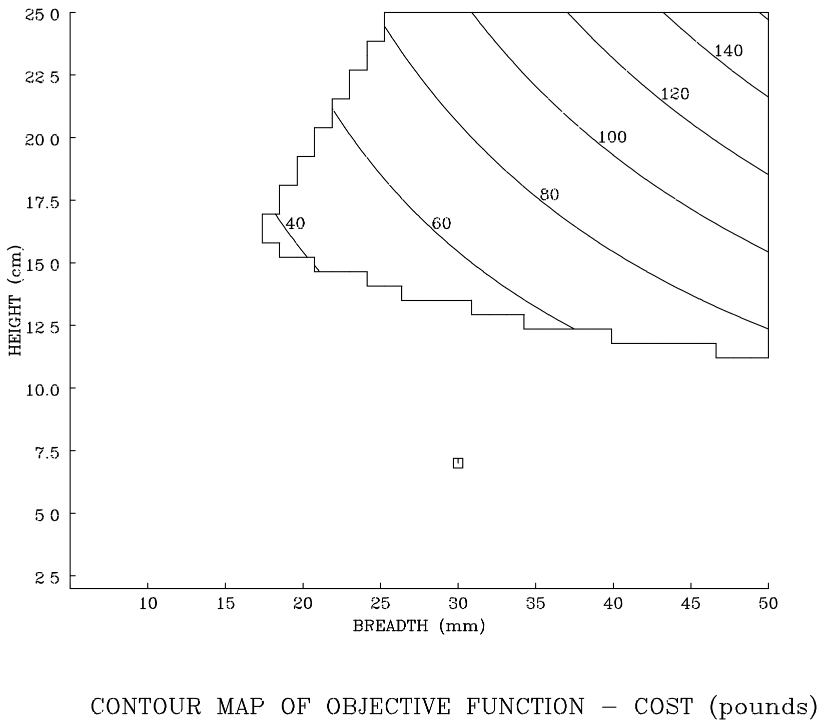



Figure 6.3 - Refined map of objective function for material type 2, high tensile steel (cost versus breadth and height).

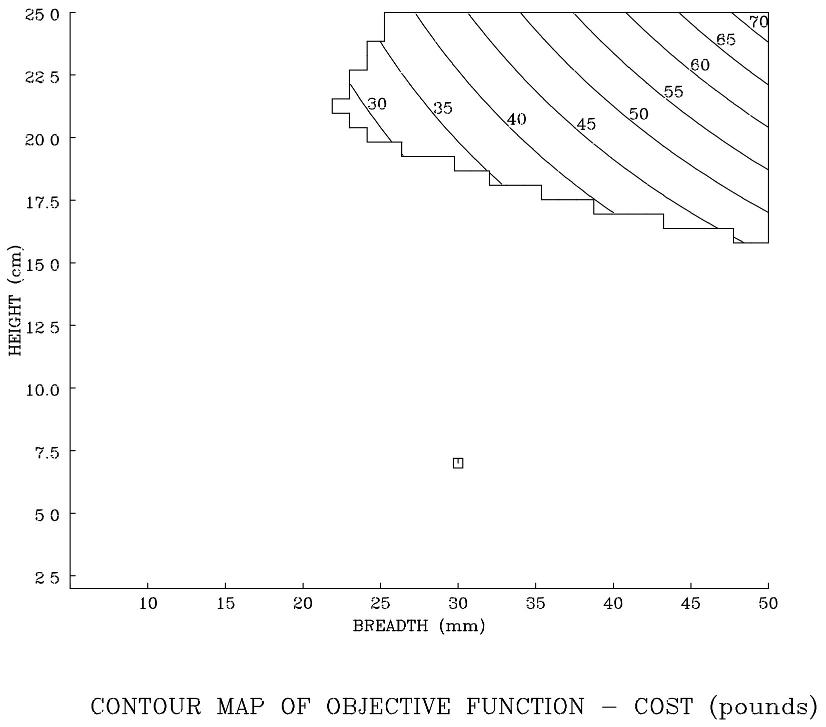



Figure 6.4 - Refined map of objective function for material type 3, aluminium alloy (cost versus breadth and height).

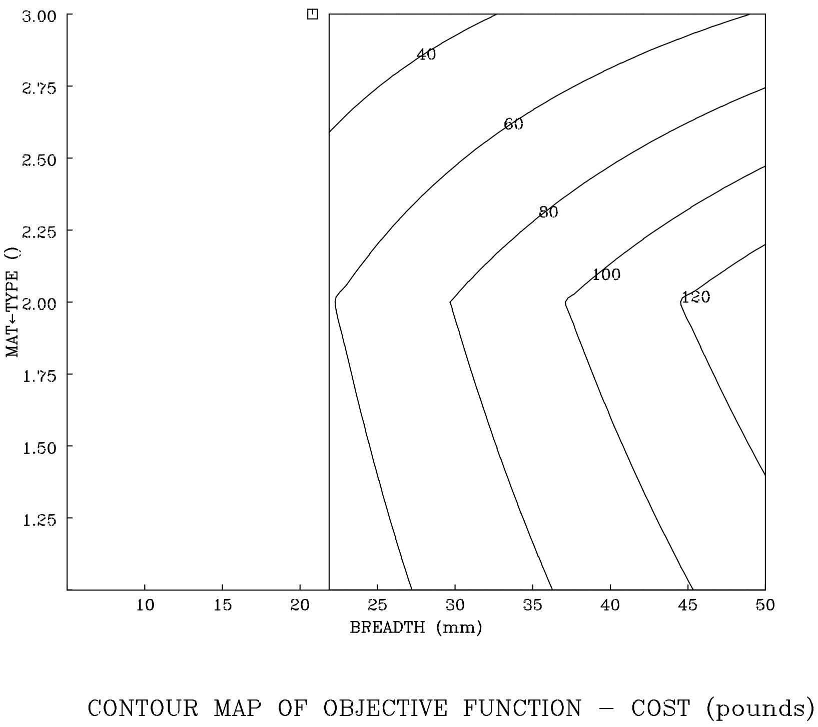
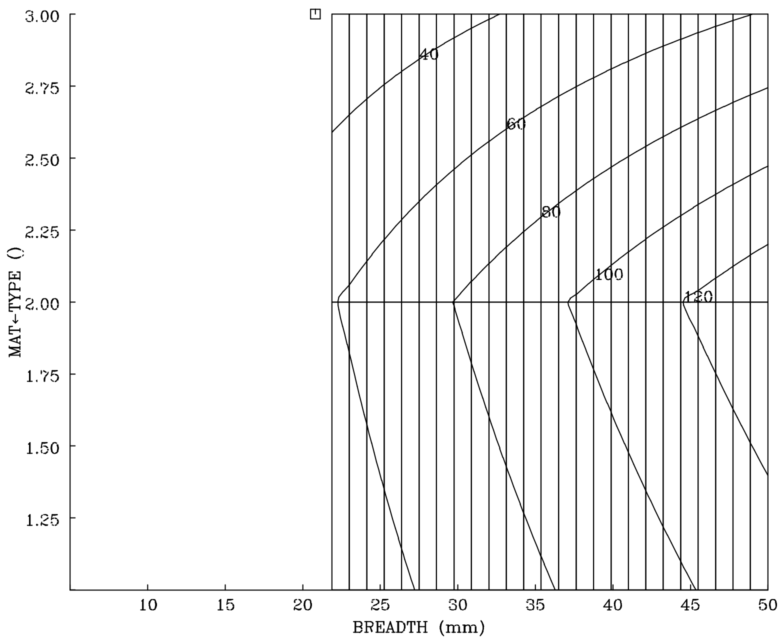



Figure 6.5 - Contour map of the objective function for fixed a height of 20.81cm (cost versus breadth and material type).




CONTOUR MAP OF OBJECTIVE FUNCTION – COST (pounds)

Figure 6.6 - Contour map of the objective function for fixed a height of 20.81cm showing element structure (cost versus breadth and material type).

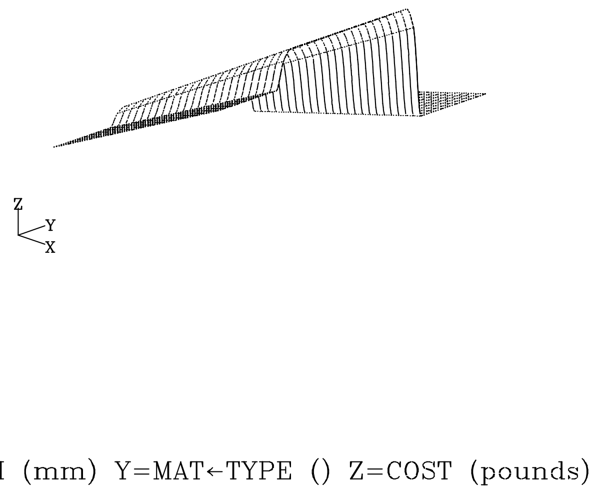



Figure 6.7 - Iso-metric projection of the objective function for fixed a height of 20.81cm (cost versus breadth and material type).

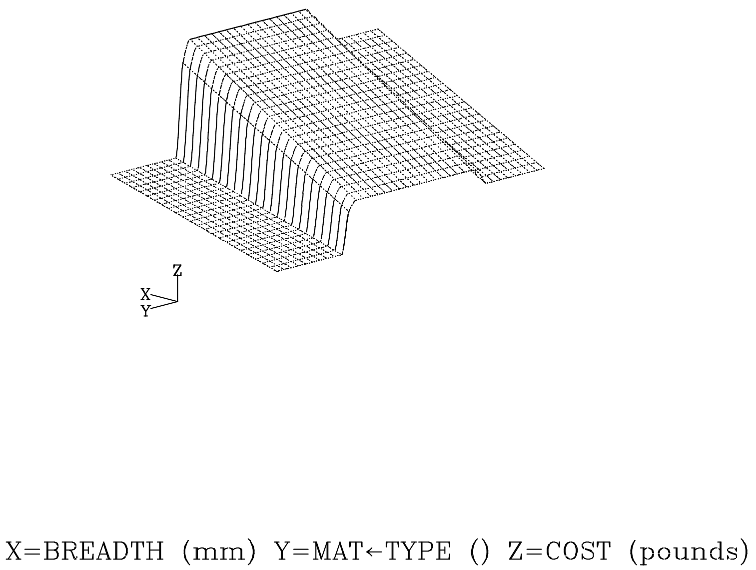



Figure 6.8 - Iso-metric projection of the objective function for fixed a height of 20.81cm with a changed view point (cost versus breadth and material type).

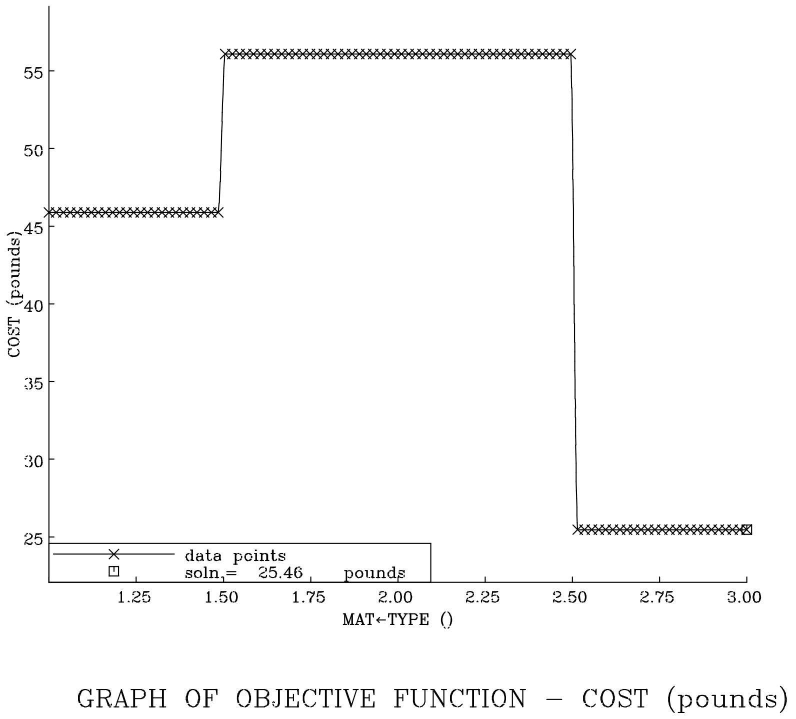



Figure 6.9 - Graph of the objective function for fixed a breadth of 20.81mm and height of 20.81cm (cost versus material type).

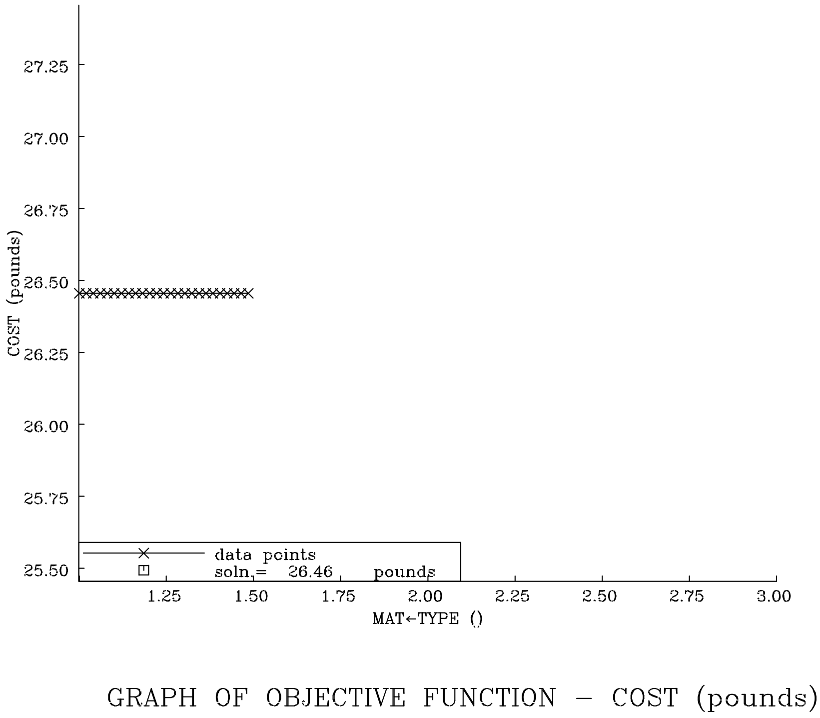



Figure 6.10 - Graph of the objective function for fixed a breadth of 15.80mm and height of 15.80cm (cost versus material type).

# CHAPTER 7

# Command Reference

## 7.1. Introduction

This chapter details all of the commands making up the seven main menus of the **OPTIONS** package illustrated in Figure 2.1 (n.b., those commands which are part of the GENDAT package and therefore intrinsic to all of the menus, are described in Appendix A). For each command the acceptable arguments are specified, followed by the sub-commands and then the help strings supplied by the program during execution. The use of the command and its variations are then set out in the following text. Note that where a group or range of integers or numbers is specified this requires the numbers to be separated by "/" symbols so that they form a single word.

## 7.2. OPTIONS Main Menu

This is the principal menu of the **OPTIONS** package from which all other menus are reached. It is entered immediately the package is started and is primarily used to check the status of the current design problem and to carry out design explorations.

### 7.2.1. Control_Optimization

Arguments: "Status", "Minimize", "Maximize", "LMinimize", "LMaximize", "RSM_model_on", "USER_model_on" or "Reset";
Sub-commands: none;
Help strings: "Dirn./type of search or reset Obj_Funct subs. and trace." and "Status, search direction, log direction, RSM on or reset.".

This command selects whether the objective function will be minimized (the default) or maximized or if the log. to base 10 will be minimized or maximized. It is also used to invoke or cancel response surface modelling of the objective function and constraints if suitable data is available in the OPT_TRACE and OPT_CTRACE data-base variables If Response Surface Modelling is to be invoked the user is subsequently prompted for RSM codes. For the objective function this should be set to 1.0 if a Shepard response surface model should be constructed, 2.1 to 2.4 for a Radial Basis Function model (2.1 for linear functions, 2.2 for thin plate splines, 2.3 for cubic splines and 2.4 for cubic spline corrections to a linear trend line with optional regression), 3.1 to 3.6 for a polynomial regression model (3.1 for mean, 3.2 for first order, 3.3 for first order plus squares, 3.4 for first order plus products (cross-terms), 3.5 for all second order and 3.6 for second order plus cubes), 4.1 for a Stochastic Process Model, 4.2 for the root mean square error of the Stochastic Process Model, 4.3 for the expected improvement of the Stochastic Process Model (4.31 gives constrained expected improvement using a constraint SPM, 4.32 gives probability of feasibility using a constraint SPM and 4.33 gives probability of improvement) and 4.4 for a twin objective Pareto front Stochastic Process Model approach (4.41 gives a model based on the combined probability of improvement, including augmentations to the front, 4.42 is combined probability of improvement but requiring all new points to dominate a memeber of the old front, 4.43 uses an average between these two measures while 4.44 to 4.46 are similar but based on the expected improvement rather than the probability of improvement). A code of zero leads to the user's code being invoked as normal, i.e., equivalent to not having the RSM turned on. For the constraints the code should be set to 1.0 if a Shepard response surface model should be constructed and used for each of the constraints and their limits in the problem, 2.1 to 2.4 if Radial Basis Function models are to be used (2.1 for linear functions, 2.2 for thin plate splines, 2.3 for cubic splines and 2.4 for cubic spline corrections to linear trend lines with optional regression) and 3.1 to 3.6 for polynomial regression models (3.1 for mean, 3.2 for first order, 3.3 for first order plus squares, 3.4 for first order plus products (cross-terms), 3.5 for all second order and 3.6 for second order plus cubes). 4.1 for a Stochastic Process Model of the first constraint and full (3.6) polynomial regression for the rest, 4.2 for the root mean square error of the Stochastic Process Model of the first constraint and full (3.6) polynomial regression for the rest.

This facility is described in a separate chapter. The default action is to show what action is currently in force. This selection *must* be made each time the program is started if *non-default* action is required as this information is not stored in the data-base files. The reset option turns off response surface modelling, objective function tracing on maps and, if the objective function is an array, resets the element used to unspecified allowing it to then be re-specified when the next action using OPTFUN is carried out.

### 7.2.2. Evaluate_Obj_Funct

Arguments: none;
Sub-commands: none;
Help strings: "Loop through the problem once with the current data.".

This command simply causes the program to loop once through the user supplied routines OPTPRE, OPTFUN, OPTCON and OPTPST. This allows for the objective function and constraints to be evaluated under manual control. This option requires the current design data-base to be correctly set-up and for it to match the user supplied code. The code must correctly evaluate the objective function within OPTFUN and all the active constraints within OPTCON.

### 7.2.3. Plot_Obj_Funct

Arguments: "Auto_Scale", "Manual_Scale", "Update" or "End_Plot";
Sub-commands: none;
Help strings: "Plot a contour map of the objective function." and "Method for fixing plot range, or just update data point.".

This command allows the objective function to be mapped and graphs, contour maps or iso-metric views to be produced. If the objective is a function of more than two variables the user must select those to be varied. The variations will span the range defined for the variables or, alternatively, a user specified range. The number of steps will typically be around 10 in auto-scaling mode. When a map has been produced the update option will place a marker on it at the current trial vector position and, if required, turn graphical tracing on. When graphical tracing is enabled the optimizers draw paths on the map as the optimization proceeds which forms a highly useful diagnostic and monitoring facility. The "Control_Optimization Reset" command will turn this tracing off again. If the update option is used before a map has been drawn a graph of objective function versus evaluation number can be produced as the optimization progresses. This allows a graphical form of monitoring to be provided. When using this feature the user must decide the likely range over which the objective function will vary and the maximum number of evaluations to be shown. Multiple optimization runs can be placed on such plots.

The end plot option closes all graphics devices, flushing any output buffers and should always be used when producing hardcopy. It has the same effect as the "Picture_Control End_Group" command in the graphics menu.

### 7.2.4. Misc_Optimizers

Arguments: "100" or a number;
Sub-commands: "Off" or a number;
Help strings: "Optimize the current problem with one of the misc. routines.", "The maximum number of iterations allowed (default 100)." and "The frequency of monitoring required (neg for pauses).".

This command transfers control to the miscellaneous optimizers menu to allow one of the methods to be applied to the current problem.

### 7.2.5. OPTIVAR_Optimizers

Arguments: "100" or a number;
Sub-commands: "Off" or a number;
Help strings: "Optimize the current problem with OPTIVAR routines.", "The maximum number of iterations allowed (default 100)." and "The frequency of monitoring required (neg for pauses).".

This command transfers control to the OPTIVAR menu to allow one of the OPTIVAR optimizers to be applied to the current problem.

### 7.2.6. Schwefel_Optimizers

Arguments: "500" or a number;
Sub-commands: "Off" or a number;
Help strings: "Optimize the current problem with Schwefel routines.", "The maximum number of iterations allowed (default 100)." and "The frequency of monitoring required (neg for pauses).".

This command transfers control to the Schwefel library menu to allow one of the methods provided in the Schwefel library to be applied to the current problem.

### 7.2.7. GAlg_Optimization

Arguments: "500" or a number;
Sub-commands: "Off" or a number;
Help strings: "Optimize the current problem with a Genetic Algorithm.", "The maximum number of iterations allowed (default 500)." and "The frequency of monitoring required (neg for pauses).".

This command invokes the genetic algorithm optimizer.

### 7.2.8. SAnn_Optimization

Arguments: "500" or a number;
Sub-commands: "Off" or a number;
Help strings: "Optimize the current problem with Simulated Annealing.", "The maximum number of iterations allowed (default 500)." and "The frequency of monitoring required (neg for pauses).".

This command invokes the simulated annealing optimizer.

### 7.2.9. EProg_Optimization

Arguments: "500" or a number;
Sub-commands: "Off" or a number;
Help strings: "Optimize the current problem with Evolutionary Programming.", "The maximum number of iterations allowed (default 500)." and "The frequency of monitoring required (neg for pauses).".

This command invokes the evolutionary programming optimizer.

### 7.2.10. EStrategy_Optimization

Arguments: "500" or a number;
Sub-commands: "Off" or a number;
Help strings: "Optimize the current problem with Evltn. Strategy Algorithm.", "The maximum number of iterations allowed (default 500)." and "The frequency of monitoring required (neg for pauses).".

This command invokes the evolutionary strategy optimizer.

### 7.2.11. Show_Optimization

Arguments: none;
Sub-commands: none;
Help strings: "Display a summary of the optimization status.".

This command displays the current status of the optimization, detailing the number of steps used by the last used method, the design vector and limits, constraints and limits together with the objective function and the type of optimization currently being carried out.

### 7.2.12. Vector_Order

Arguments: a number;
Sub-commands: "Variables", "Equalities", "Inequalities", "Ranges", "Parameters", "Intermediates" or "Tertiaries";
Help strings: "Select the size of problem vectors (if dflt vectors in use).", "The length of the vector being defined." and "The (default) problem vector to be defined.".

This command adjusts the dimensions of the data-base variables "VARS", "EQC", "INEQC", "RNGC", "PARAMETERS", "INTERMED" and "TERTIARY" to be (1,1,n) where n is the argument, selecting which variables to change according to the selected sub-command. When changing "VARS", "EQC", "INEQC" or "RNGC" it also adjusts their upper, lower or equality limits as well. If the variables do not exist an error is generated. This option speeds up the manipulation of the optimization variables if these names are adopted in the user supplied problem code and data-base.

#### 7.2.13. Graphics

Arguments: "Default_Style", "style/opt_trace" or a string;
Sub-commands: none;
Help strings: "General purpose graph preparation facility." and "The file name for the graphics style file.".

This command transfers control to the graphics menu which can be used for the production and manipulation of plots.

#### 7.2.14. Main_System

Arguments: none;
Sub-commands: none;
Help strings: "Return to the main system or program.".

This command leaves the **OPTIONS** system and is available for installations where the package is included as a set of subroutines to a bigger design code, allowing the user to revert control to that code. In the absence of such a code it is equivalent to the "Quit_to_System" command.

### 7.3. Misc. Optimizers Menu

This menu is used to select from a miscellaneous collection of optimizers along with their major control options. Some of these methods do not treat range constraints on the design variables as mandatory so the menu also includes an option to enforce this restriction.

#### 7.3.1. Control_Option

Arguments: "Advisory_DV_Constraints" or "Mandatory_DV_Constraints";
Sub-commands: "No_Internal_Printout" or a number;
Help strings: "Select additional control facilities.", "The type of design variable constraints for penalty methods." and "Internal print-out frequency.".

This command allows the user to enforce range constraints on the design variables since some of the methods in the miscellaneous collection only treat these as advisory. It also allows the internal print options of the routines to be enabled to give full diagnostics. A print-out frequency of 0 just gives information on starting the optimizer, otherwise information is produced more frequently, with bigger numbers producing more output.

#### 7.3.2. U1_Optimizer

Arguments: a number;
Sub-commands: a number;
Help strings: "Optimize the current problem with User Defined Algorithm 1.", "First user defined parameter." and "Second user defined parameter.".

This command invokes the first user defined optimizer.

#### 7.3.3. U2_Optimizer

Arguments: a number;
Sub-commands: a number;
Help strings: "Optimize the current problem with User Defined Algorithm 2.", "First user defined parameter." and "Second user defined parameter.".

This command invokes the second user defined optimizer.

#### 7.3.4. NAg_Optimizer

Arguments: none;
Sub-commands: none;

Help strings: "Optimize the current problem with NAg routines.".

This command invokes the NAg optimizers on the current problem. Normally this implies the routine E04UCF although versions of the package using the older routines E04CGF, E04JAF and E04UAF can be supplied.

#### 7.3.5. BClimb_Optimizer

Arguments: none;
Sub-commands: none;
Help strings: "Optimize the current problem with Bit Climbing Algorithm.".

This command invokes the bit climbing optimizer.

#### 7.3.6. DHClimb_Optimizer

Arguments: none;
Sub-commands: none;
Help strings: "Optimize the current problem with Dyn. Hill Climb. Algorithm.".

This command invokes the dynamic hill climbing optimizer.

#### 7.3.7. PBIL_Optimizer

Arguments: none;
Sub-commands: none;
Help strings: "Optimize the current problem with PBIL Algorithm.".

This command invokes the population based incremental learning optimizer.

#### 7.3.8. Num_Rcp_Optimizer

Arguments: "One_Pass_External" or "Fiacco-McCormick";
Sub-commands: "Powell", "Polak-Ribiere", "Fletcher-Reeves", "Broyden-Fletcher", "GN_Polak-Ribiere" or "GN_Fletcher-Reeves";
Help strings: "Optimize the current problem with Num. Recipes Algorithm.", "The penalty function to be used." and "Method to use.".

This command invokes the optimizers described in "Numerical Recipes: The Art of Scientific Computing"[4] It allows use of that book's routines called DFPMIN (Broyden- Fletcher-Goldfarb-Shanno variant of Davidon-Fletcher-Powell minimization), FRPRMN (Fletcher-Reeves-Polak-Ribiere minimization) and POWELL (Powell method discarding the direction of largest decrease). It also allows use of two variants on the conjugate gradient methods described by Gilbert, J.C. and Nocedal, J. (1992). "Global Convergence Properties of Conjugate Gradient Methods", SIAM Journal on Optimization, Vol. 2, pp. 21-42. These are accessed by the "GN_" prefixed commands. Both the routines DFPMIN and FRPRMN require information on objective function gradients. When needed these are synthesized numerically by finite differencing with a fixed small step length. (N.b., by using the OPT_GRADNT feature it is possible to use direct gradient calculations in place of these finite difference calculations.) Step lengths are controoled by the data-base variable "NRP_DX" which has a default of 1.0E-7.

As described in the references the routines used are for dealing with unconstrained problems. Here constraints are handled by the use of penalty functions. By default this is a one pass external function which is very fast but risks stalling the searches. In such cases restart the search using the Fiacco-McCormick multi-pass function which is more robust but slower, n.b., when using this slower method it is important to check that the final penalty achieved is sufficient to ensure feasible constraints, limited numbers of function calls commonly prevent this from being achieved.

#### 7.3.9. DoE_Optimizer

Arguments: "Random", "LP_tau", "Central_composite", "FCC_factorial", "Latin_hypercube", "Cell-based_Latin_Hypercube" or "Candidate_list";
Sub-commands: none;
Help strings: "Optimize the current problem with DoE Algorithm." and "The search pattern used to construct the trial vectors.".

This command invokes the Design of Experiments optimizer with either random design points, points fixed using $LP_\tau$ methods, a central composite full factorial design augmented with $LP_\tau$ points, a face centred cubic full factorial design augmented with $LP_\tau$ points, a Latin hypercube design (with two variants), or a list of candidate points taken from the DOE_TRACE array (and modified to exclude infeasible points if such information is available in the DOE_CTRACE array). If the candidate list contains more valid points than needed for the optimization the user is asked if a small search over the available points in the list should be performed so as to improve the inter-point Euclidean spacings.

### 7.3.10. Test_DoE_Optimizer

Arguments: "Random", "LP_tau", "Central_composite", "FCC_factorial", "Latin_hypercube", "Cell-based_Latin_Hypercube" or "Candidate_list";
Sub-commands: "Skip_constraint_calcs" or "Run_constraint_calcs";
Help strings: "Test the DoE Algorithm without calling the current problem.", "The search pattern used to construct the trial vectors." and "Choose if full constraint calculations are to be run.".

This command invokes the Design of Experiments optimizer with either random design points, points fixed using $LP_\tau$ methods, a central composite full factorial design augmented with $LP_\tau$ points, a face centred cubic full factorial design augmented with $LP_\tau$ points, a Latin hypercube design (with two variants), or a list of candidate points taken from the DOE_TRACE array (and modified to exclude infeasible points if such information is available in the DOE_CTRACE array). However, the user's objective function code is not called when this command is used while the user's constraint codes is only called if explicitly asked for in the command. This allows the user to see what points would be used if the normal "DoE_Optimizer" command where used (by examining the contents of the OPT_TRACE and OPT_CTRACE data-base variables after executing the command). After this routine has been run the OPT_TRACE and OPT_CTRACE trace arrays may be modified and renamed by the user before the "Candidate_list" option of the "DoE_Optimizer" command is invoked to run them as DOE_TRACE and DOE_CTRACE.

### 7.3.11. Optimizer_Menu

Arguments: none;
Sub-commands: none;
Help strings: "Return to the main optimizer menu.".

This command simply reverts control to the main **OPTIONS** menu, aborting the miscellaneous optimization.

## 7.4. OPTIVAR Menu

This menu is used to select from the various optimizers within the OPTIVAR package along with their major control options. Some of these methods do not treat range constraints on the design variables as mandatory so the menu also includes an option to enforce this restriction. The intrinsic OPTIVAR diagnostics may also be enabled during optimization, to allow for full monitoring of the method selected.

### 7.4.1. Adrans_Optimizer

Arguments: "One_Pass_External", "Fiacco-McCormick", "Powell" or "Schuldt";
Sub-commands: "128" or a number;
Help strings: "Adaptive Random Search.", "The penalty function to be used." and "Seed for random number sequence (default 128).".

This command invokes the ADRANS optimizer from the OPTIVAR suite.

### 7.4.2. David_Optimizer

Arguments: "One_Pass_External", "Fiacco-McCormick", "Powell" or "Schuldt";
Sub-commands: "1.0E-5" or a number;
Help strings: "Davidon-Fletcher-Powell Method.", "The penalty function to be used." and "Convergence criterion on objective (default 1.0E-5).".

This command invokes the DAVID optimizer from the OPTIVAR suite.

### 7.4.3. Fletch_Optimizer

Arguments: "One_Pass_External", "Fiacco-McCormick", "Powell" or "Schuldt";
Sub-commands: "1.0E-4" or a number;
Help strings: "Fletcher"s 1972 Method.", "The penalty function to be used." and "Convergence criterion on variables (default 1.0E-4).".

This command invokes the FLETCH optimizer from the OPTIVAR suite.

### 7.4.4. Jo_Optimizer

Arguments: "One_Pass_External", "Fiacco-McCormick", "Powell" or "Schuldt";
Sub-commands: "1.0E-4" or a number;
Help strings: "Jacobson and Oksman Method.", "The penalty function to be used." and "Convergence criterion on variables (default 1.0E-4).".

This command invokes the JO optimizer from the OPTIVAR suite.

### 7.4.5. PDS_Optimizer

Arguments: "One_Pass_External", "Fiacco-McCormick", "Powell" or "Schuldt";
Sub-commands: "1.0E-4" or a number;
Help strings: "Powell"s Direct Search Method.", "The penalty function to be used." and "Convergence criterion on variables (default 1.0E-4).".

This command invokes the PDS optimizer from the OPTIVAR suite. It should not be used with the one-pass external penalty function due to a coding bug.

### 7.4.6. Seek_Optimizer

Arguments: "One_Pass_External", "Fiacco-McCormick", "Powell" or "Schuldt";
Sub-commands: "0.01" or a number;
Help strings: "Hooke and Jeeves Direct Search.", "The penalty function to be used." and "Min. step length stopping criterion (default 0.01).".

This command invokes the SEEK optimizer from the OPTIVAR suite.

### 7.4.7. Simplx_Optimizer

Arguments: "One_Pass_External", "Fiacco-McCormick", "Powell" or "Schuldt";
Sub-commands: "1.0E-4" or a number;
Help strings: "Simplex Method.", "The penalty function to be used." and "Convergence criterion on variables (default 1.0E-4).".

This command invokes the SIMPLX optimizer from the OPTIVAR suite.

### 7.4.8. Approx_Optimizer

Arguments: "0.1" or a number;
Sub-commands: "20" or a number;
Help strings: "Method of Successive Linear Approximation.", "Fraction of range limiting step lengths (default 0.1)." and "Max. number of simplex iterations (default 4*M+10).".

This command invokes the APPROX optimizer from the OPTIVAR suite.

### 7.4.9. Random_Optimizer

Arguments: "5" or a number;
Sub-commands: "4" or a number,
Help strings: "Random Exploration with Shrinkage.", "No. of points retained per iteration (default 10*N/NSHRIN)." and "Shrinkage factor (default 4).".

This command invokes the RANDOM optimizer from the OPTIVAR suite.

### 7.4.10. Reduce_Optimizer

Arguments: "0.0001" or a number;
Sub-commands: "1.0E-6" or a number;
Help strings: "Reduced Gradient Method.", "Max. infeasibility on consts. and bounds (default 0.0001)." and "Line search stopping criterion (default 1.0E-6).".

This command is designed to invoke the REDUCE optimizer from the OPTIVAR suite but is currently not implemented.

### 7.4.11. Control_Option

Arguments: "Advisory_DV_Constraints" or "Mandatory_DV_Constraints";
Sub-commands: "No_Internal_Printout" or a number;
Help strings: "Select additional control facilities.", "The type of design variable constraints for penalty methods." and "OPTIVAR internal print-out frequency (IPRINT).".

This command allows the user to enforce range constraints on the design variables since some of the methods in the OPTIVAR suite only treat these as advisory. It also allows the internal print options of the OPTIVAR library to be enabled to give full diagnostics. A print-out frequency of -1 gives just the header information, otherwise information is produced every nth cycle where n is the specified frequency; the default is zero.

### 7.4.12. Optimizer_Menu

Arguments: none;
Sub-commands: none;
Help strings: "Return to the main optimizer menu.".

This command simply reverts control to the main **OPTIONS** menu, aborting the OPTIVAR optimization.

## 7.5. Schwefel Menu

This menu is used to select from the various optimizers within the Schwefel library along with their major control options.

### 7.5.1. Fibonacci_Optimizer

Arguments: "One_Pass_External" or "Fiacco-McCormick";
Sub-commands: none;
Help strings: "Repeated application of a 1-d Fibonacci search." and "The penalty function to be used.".

This command invokes the FIBO optimizer from the Schwefel suite.

### 7.5.2. Golden_Sect_Optimizer

Arguments: "One_Pass_External" or "Fiacco-McCormick";
Sub-commands: none;
Help strings: "Repeated application of a 1-d Golden Section search." and "The penalty function to be used.".

This command invokes the GOLD optimizer from the Schwefel suite.

### 7.5.3. Lagrange_Int_Optimizer

Arguments: "One_Pass_External" or "Fiacco-McCormick";
Sub-commands: none;
Help strings: "Repeated application of a 1-d Lagrangian interpolation." and "The penalty function to be used.".

This command invokes the LAGR optimizer from the Schwefel suite.

### 7.5.4. Hooke&Jeeves_Optimizer

Arguments: "One_Pass_External" or "Fiacco-McCormick";
Sub-commands: none;

Help strings: "Hooke & Jeeves direct search." and "The penalty function to be used.".

This command invokes the HOJE optimizer from the Schwefel suite.

### 7.5.5. Rosenbrock_Optimizer

Arguments: none;
Sub-commands: none;
Help strings: "Rosenbrock rotating co-ordinate search.".

This command invokes the ROSE optimizer from the Schwefel suite.

### 7.5.6. DSCG_Optimizer

Arguments: "One_Pass_External" or "Fiacco-McCormick";
Sub-commands: none;
Help strings: "Davies, Swann and Campey strategy with Gram-Schmidt orthog." and "The penalty function to be used.".

This command invokes the DSCG optimizer from the Schwefel suite.

### 7.5.7. DSCP_Optimizer

Arguments: "One_Pass_External" or "Fiacco-McCormick";
Sub-commands: none;
Help strings: "Davies, Swann and Campey strategy with Palmer orthog." and "The penalty function to be used.".

This command invokes the DSCP optimizer from the Schwefel suite.

### 7.5.8. Powell_Optimizer

Arguments: "One_Pass_External" or "Fiacco-McCormick";
Sub-commands: "default_conv" or "alternate_conv";
Help strings: "Powell's strategy of conjugate directions.", "The penalty function to be used." and "Convergence criterion.".

This command invokes the POWE optimizer from the Schwefel suite.

### 7.5.9. DFPS_Optimizer

Arguments: "One_Pass_External" or "Fiacco-McCormick";
Sub-commands: "50%_improvement" or a number;
Help strings: "Davidon, Fletcher, Powell strategy.", "The penalty function to be used." and "Expected objective function value at the solution point.".

This command invokes the DFPS optimizer from the Schwefel suite.

### 7.5.10. Simplex_Optimizer

Arguments: "One_Pass_External" or "Fiacco-McCormick";
Sub-commands: "MCalls_over_25NVars" or a number;
Help strings: "Simplex strategy of Nelder and Meade.", "The penalty function to be used." and "Number of iterations before starting convergence tests.".

This command invokes the SIMP optimizer from the Schwefel suite.

### 7.5.11. Complex_Optimizer

Arguments: none;
Sub-commands: none;
Help strings: "Complex strategy of M.J. Box.".

This command invokes the COMP optimizer from the Schwefel suite.

### 7.5.12. 2M_ES_Optimizer

Arguments: "1" or a number;
Sub-commands: "0.85" or a number
Help strings: "Two-membered evolution strategy", "Factor on successes per trial for step refinement." and "Step size refinement ratio.".

This command invokes the EVOL optimizer from the Schwefel suite. The Factor on successes per trial for step refinement is an auxiliary quantity used in step size management (Schwefel's parameter LR). Its normal value is one which implies that step sizes are adjusted so that, on average, one success (improvement in the value of the objective function) is obtained in five times this parameter number of trials (objective function calls), i.e., if this is changed to two, on average one success will be obtained in every ten trials and so on. The step size refinement ratio is used to adjust the random step sizes made by the strategy (Schwefel's parameter SN). This adjustment is carried out after every $LR \times N$ trials, where $N$ is the number of variables in the problem.

### 7.5.13. MM_ES_Optimizer

Arguments: "comma" or "plus";
Sub-commands: "333" or a number;
Help strings: "Multi-membered evolution strategy.", "Comma or plus version of the Evolution Strategy." and "Recombination type - three integers in the range 1-5.".

This command invokes the KORR optimizer from the Schwefel suite. A "comma" strategy implies that new generations are selected only from children while a "plus" strategy allows new generations to be derived from both parents and children. The recombination type (Schwefel's parameter IREKOM) is a three digit integer in the range 111 to 555, where the first digit encodes the type of recombination for the variables, the second the type of recombination for the step-sizes, and the third the type of recombination for the correlation angles. The types used are: 1 for no recombination; 2 for discrete recombination of pairs of parents; 3 for intermediate recombination of pairs of parents; 4 for discrete recombination of all parents and 5 for intermediate recombination of all parents in pairs.

### 7.5.14. Optimizer_Menu

Arguments: none;
Sub-commands: none;
Help strings: "Return to the main optimizer menu.".

This command simply reverts control to the main **OPTIONS** menu, aborting the Schwefel optimization.

## 7.6. Graphics Menu

The graphics menu allows for the manipulation of graphical output. It may be accessed directly from the main **OPTIONS** menu but is automatically invoked when mapping objective functions, either in full form or a reduced version which simply allows for the selection and initialization of the desired graphics device. Two example macro command files, "testgr" and "test3d", together with their associated data and style files, are included with the package and demonstrate most of the facilities available from this menu.

### 7.6.1. Select_Device

Arguments: "SUN", "XTERM", "LASERW", "POSTSCRIPT", "HPGL", "PIC_TROFF", "SPECIFIED_DEVICE";
Sub-commands: none;
Help strings: "Select the type of graphics terminal in use." and "A device name".

This command allows the current graphics device to be changed. The command is not effective until the current picture is completed with the "Picture_Control End_Group" command. The devices available depend on the installation details.

### 7.6.2. Picture_Control

Arguments: "Group_Off", "End_Group", "New_Graph", "Open_Device" or a group of integers;
Sub-commands: none;

Help strings: "Control the graph grouping etc." and "Group layout or control parameter.".

This command allows the graphics image to be sub-divided into a number of plots, which implies closing the existing plot if one is open. Few devices have a resolution capable of dealing with more than 3x3 plots per page. The "End_Group" argument also closes the graphics device allowing a new one to be selected.

#### 7.6.3. Main_System

Arguments: none;
Sub-commands: none;
Help strings: "Return to the main system or program.".

This command returns control to the routine that called the graphics facilities, usually the previous menu level (in stand-alone graphics uses this may mean the operating system).

#### 7.6.4. Define_Curve

Arguments: none;
Sub-commands: none;
Help strings: "Define the data and style for curves.".

This command enters the curve definition menu to enable the features of each curve within a plot to be set.

#### 7.6.5. Captions

Arguments: "Primary", "Secondary", "X-Axis" or "Y-Axis";
Sub-commands: "B", "T", "L", "H" or "Off";
Help strings: "Input the title and axes captions.", "Caption type - "Primary", "Secondary", "X-axis" or "Y-axis"." and "Primary title vertical position or "Off".".

This command allows captions to be added to the page, plot or axes. "B" and "T" fix the overall captions with respect to the plot while "L" and "H" fix them with respect to the page. This distinction is most relevant when multiple plots are placed on one page using the grouping controls. If the caption contains a string of the form "‘VARNAM(X,Y,Z)‘" this is replaced by the appropriate numeric value in the caption (note the use of *back* quotes). If the string contains the character "@" this changes the output into the mathematical font or back.

#### 7.6.6. Key_Position

Arguments: "H", "L", "B", "T" or "Off";
Sub-commands: "R", "L", "C", "E" or "W";
Help strings: "Select the key box position.", "Key box vertical position or "Off"." and "Key box horizontal position.".

This command allows key boxes to be added to the page or plot. "R", "L", "C", "B" and "T" fix the overall captions with respect to the plot while "E", "W", "L" and "H" fix them with respect to the page. This distinction is most relevant when multiple plots are placed on one page using the grouping controls.

#### 7.6.7. X-axis_Set

Arguments: "Linear", "Logarithmic" or "Polar";
Sub-commands: ,"Auto-scale" or a group of numbers;
Help strings: "Select the X-axis type and range.", "Type of scales ("Linear", "Logarithmic" or "Polar")." and "Scale range as "a/b" - a,b real numbers or "Auto-scale".".

This command selects the X-axis type and range. If polar axes are selected this controls both X- and Y-axes.

#### 7.6.8. Y-axis_Set

Arguments: "Linear", "Logarithmic" or "Polar";
Sub-commands: "Auto-scale" or a group of numbers;
Help strings: "Select the Y-axis type and range.", "Type of scales ("Linear", "Logarithmic" or "Polar")." and "Scale range as "a/b" - a,b real numbers or "Auto-scale".".

This command selects the Y-axis type and range. If polar axes are selected this controls both X- and Y-axes.

### 7.6.9. Z-axis_Set

Arguments: "Auto-scale" or a group of numbers;
Sub-commands: none;
Help strings: "Select the Z-axis range." and "Scale range as "a/b" - a,b real numbers or "Auto-scale".".

This command selects the Z-axis range. This is only used if iso-metric plots are being produced. The Z-axis is always linear.

### 7.6.10. File_Options

Arguments: the last used style file name or a character string;
Sub-commands: "Store" or "Recall";
Help strings: "Store or recall the graphics options from file.", "A file name ( < 25 characters ), default is last used name." and ""Store"(default) or "Recall".".

This command allows the current plot control settings to be stored or retrieved. This is distinct from the data-base itself and includes sufficient information to re-create plots if the relevant data-base is also restored. It does not deal with group information or the choice of graphics device.

### 7.6.11. Draw

Arguments: "All", "Axes", "Key_Box" or a group of integers;
Sub-commands: ,"Plot", "List" or a group of integers;
Help strings: "Draw or list graph elements.", "First and last curve numbers, "All", "Axes" or "Key"." and "Plot curves, list curve details or key entry plot range.".

This command produces the desired picture. When used in its default mode this creates and completes one complete plot. A subsequent "Draw" will then start a new plot, either as part of a group or on a fresh page (which means overwriting the previous plot on some graphics devices). Alternatively plots may be built up manually, by drawing curves individually or in groups. These may be preceded with a manually fixed set of axes, otherwise the first drawn curve will fixed the scales. Curves can then be drawn on the axes up until the plot is ended, either with a "Draw All" or a "Picture_Control" command. If a "Draw Axes" command is given before a plot is ended a second set of Y-axes are drawn allowing for graphs with twin scales. Subsequent curves are drawn based on the second set of axes drawn. The key box may be drawn at any time but only contains entries for those curves currently on the plot. The "Draw All List" options displays information on each curve in turn: this is where the results of polynomial regression fits may be studied as well as the numbers of points making up curves, their types, etc. If a polynomial curve fit has been used, it also gives details of the accuracy of the fit: it gives the sum of the squares of the errors as "Delta sqd", the standard deviation as "Std Dev" and the biased sum of the squares of the errors as "Sigma sqd". This last quantity should be minimized to select the correct curve fitting order, according to the null hypothesis.

### 7.6.12. Contour_Control

Arguments: "Labels", "Edges", "Elements", "Gaps" or "List";
Sub-commands: "On" or "Off";
Help strings: "Control appearance of contour maps.", "Feature to be turned on or off." and ""On" or "Off".".

This command allows the details of contour maps to be changed. Each contour map is considered to form one curve allowing multiple maps to be laid on top of each other if desired. Contour maps are drawn when the individual curve details have sufficient information to allow this; in particular the "Z-data_name" needs to be set, see the "Define_Curve" menu.

### 7.6.13. Surface_Control

Arguments: "Off", "Outline", "Contour", "X-Hatch" or "List";
Sub-commands: "20/1/15", "Default" or a group of integers;
Help strings: "Control appearance of iso-metric surfaces.", "Type of surface plot required." and "Mesh interval, view corner and angle (im/ic/ia) or default.".

This command allows the details of iso-metric plots to be changed. Only one plot may be displayed at any one time and normally this is based on the data used for plotting curve one as a contour map.

#### 7.6.14. Transform_Data

Arguments: none;
Sub-commands: none;
Help strings: "Transform the data within the data-base.".

This command transfers control to the Transform Data menu.

### 7.7. Define Curve Menu

Plots produced by the graphics package may have many curves placed on each graph. The source data and style of these may differ from curve to curve and up to ten such curve setups may be defined at any one time. These settings are manipulated from this menu.

#### 7.7.1. X-data_Name

Arguments: "XDATA" or a character string;
Sub-commands: "First_Curve", "All_Curves" or a number;
Help strings: "Define the names for the X-data ("*" to turn curve off).", "A valid variable name in the current data structure." and "The curves for which this variable will be used.".

This command selects the variable name to form the X-data for the designated curve.

#### 7.7.2. Y-data_Name

Arguments: "YDATA" or a character string;
Sub-commands: "First_Curve", "All_Curves" or a number;
Help strings: "Define the names for the Y-data.", "A valid variable name in the current data structure." and "The curves for which this variable will be used.".

This command selects the variable name to form the Y-data for the designated curve.

#### 7.7.3. Z-data_Name

Arguments: ""not_set"" or a character string;
Sub-commands: "First_Curve", "All_Curves" or a number;
Help strings: "Define the names for the Z-data.", "A valid variable name in the current data structure." and "The curves for which this variable will be used.".

This command selects the variable name to form the Z-data for the designated curve. This is only relevant when contour maps or iso-metric views are to be produced. Z-data values of -0.10E+21 are taken to indicate gaps in the surface to be plotted.

#### 7.7.4. C-data_Name

Arguments: ""not_set"" or a character string;
Sub-commands: "First_Curve", "All_Curves" or a number;
Help strings: "Define the names for the contour height data.", "A valid variable name in the current data structure." and "The curves for which this variable will be used.".

This command selects the variable name to form the contour data for the designated curve. Individual contour lines are then drawn on contour maps for this curve at the levels contained in the variable.

#### 7.7.5. Plot_Control

Arguments: "On" or "Off";
Sub-commands: "First_Curve", "All_Curves" or a number;
Help strings: "Turn a curve on or off.", "Whether the curve is drawn ("On" default) or not ("Off")." and "The curves which will be affected.".

This command enables or disables individual curves. By default only the first curve is enabled when the package is started.

### 7.7.6. Range_of_Data

Arguments: "X-Data", "Y-Data", "Z-Data" or "C-Data";
Sub-commands: "First_Curve", "All_Curves" or a number;
Help strings: "Define the ranges of the data element subscripts.", "The set for detailed subscript input ("X-data", etc.)." and "The curves for which this set of ranges will be used.".

This command fixes which elements within individual data-base variables are to be used when producing curves. When used the program then prompts for three pairs of numbers which form the X, Y and Z subscript ranges to be used.

### 7.7.7. Curve_Type

Arguments: "Just_Points", "Tight_Curve", "Loose_Curve", "Straight_Lines", "Polynomial_Regression", "Histogram", "Cumulative_Histogram", "Density_Fn", "Distribution_Fn" or "User_Defined_Function";
Sub-commands: "First_Curve", "All_Curves" or a number;
Help strings: "Select the curve types.", "The method for joining points forming the curves." and "The curves for which this curve type will be used.".

This command selects the type of curve to be used for each set of data. If the type selected requires additional information, such as the order of the polynomial to be used, this is then prompted for. If it is to be used, the user defined function must be supplied when the package is compiled and linked or a default null routine is included. It should take the form of a real function called "USRFTN" with a single real argument This is then plotted over the ranges of the graph with a suitable number of points depending on the local curvature. This feature enables the user to provide a problem specific function if desired.

### 7.7.8. Line_Type

Arguments: a number;
Sub-commands: "First_Curve", "All_Curves", "Incrementally" or a number;
Help strings: "Select the line patterns (type numbers from -6 to +6).", "A number indicating the line pattern (solid, chain etc.)." and "The curves for which this line type will be used.".

This command selects the various line patterns available, which may be set for all curves in one go using the incremental option.

### 7.7.9. Pen_Type

Arguments: a number;
Sub-commands: "First_Curve", "All_Curves", "Incrementally" or a number;
Help strings: "Select the pen thickness/colour (type numbers from 0 to 8).", "A number indicating the pen type (0 for off)." and "The curves for which this pen type will be used.".

This command selects the various pen types (colours) available, which may be set for all curves in one go using the incremental option.

### 7.7.10. Symbol_Type

Arguments: "Off" or a group of integers;
Sub-commands: "First_Curve", "All_Curves", "Incrementally" or a number;
Help strings: "Select the symbol patterns (type numbers from 0 to 16).", "A symbol number, increment and pattern number (or "Off")." and "The curves for which this symbol type will be used.".

This command selects the various symbol patterns available, which may be set for all curves in one go using the incremental option. If the argument is given as a group of integers the first integer is placed next to each symbol, skipping the second integer number of points between symbols and using the pattern indicated by the third. A pattern type of 16 is a null pattern.

### 7.7.11. Caption

Arguments: "First_Curve", "All_Curves", "Incrementally" or a number;
Sub-commands: none;
Help strings: "Select the key box captions." and "The curves for which the caption will be used.".

This command allows a key box caption to be defined for a curve. If this contains a string of the form "‘VARNAM(X,Y,Z)‘" this is replaced by the appropriate numeric value in the caption (note the use of *back* quotes). If the string contains the character "@" this changes the output into the mathematical font or back.

### 7.7.12. List_Details

Arguments: "First_Curve", "All_Curves", "Incrementally" or a number;
Sub-commands: none;
Help strings: "List the details specified for curves." and "The curves for which the details will be listed.".

This command simply lists the currently set details for the selected curve or curves. If a polynomial curve fit has been specified and used, it gives details of the accuracy of the fit and the coefficients used. It gives the sum of the squares of the errors as "Delta sqd", the standard deviation as "Std Dev" and the biased sum of the squares of the errors as "Sigma sqd". This last quantity should be minimized to select the correct curve fitting order, according to the null hypothesis.

### 7.7.13. Graph_Menu

Arguments: none;
Sub-commands: none;
Help strings: "Return to the main graphics option menu.".

This command returns control to the graphics menu.

## 7.8. Transform Data Menu

The transform data menu allows access to a group of commands that may be used to manipulated data within the data-base. These are primarily aimed at vectors of numbers and allow a limited but useful group of operations. The vectors used need not be of the same dimensions: if they are not similar warnings are given. Warnings are also given if data will be over-written by any transformation.

### 7.8.1. Scale_and_Shift

Arguments: a character string;
Sub-commands: "Onto_Itself" or a character string;
Help strings: "Take Ax**2+Bx+C of data within an existing data structure.", "The name of the variable to be worked on." and "The name of the variable for the results.".

The real constants $A$, $B$ and $C$ are prompted for when this command is invoked. Any of the three constants may be set to zero.

### 7.8.2. Power_Law

Arguments: a character string;
Sub-commands: "Onto_Itself" or a character string;
Help strings: "Take power of data within an existing data structure.", "The name of the variable to be worked on." and "The name of the variable for the results.".

This command uses a formula of the form $((Bx)^A)/C$ to transform the variables. The real constants $A$, $B$ and $C$ are prompted for when this command is invoked and any of the three may be set to zero. Using $B$ and $C$ of unity and $A$ of -1 allows inverses to be formed.

### 7.8.3. Log10(Abs(Value))

Arguments: a character string;
Sub-commands: "Onto_Itself" or a character string;
Help strings: "Take Logs of data within an existing data structure.", "The name of the variable to be worked on." and "The name of the variable for the results.".

### 7.8.4. ALog10(Value)

Arguments: a character string;
Sub-commands: "Onto_Itself" or a character string;
Help strings: "Take ALogs of data within an existing data structure.", "The name of the variable to be

worked on." and "The name of the variable for the results.".

### 7.8.5. Col_Sum_Values

Arguments: a character string;
Sub-commands: "Onto_Itself" or a character string;
Help strings: "Sum cols. of a variable within an existing data structure.", "The name of the variable to be worked on." and "The name of the variable for the results.".

If the variable is summed to itself the results column is not included in the summation, which allows column totals to be easily set up and manipulated.

### 7.8.6. Row_Sum_Values

Arguments: a character string;
Sub-commands: "Onto_Itself" or a character string;
Help strings: "Sum rows of a variable within an existing data structure.", "The name of the variable to be worked on." and "The name of the variable for the results.".

If the variable is summed to itself the results row is not included in the summation, which allows row totals to be easily set up and manipulated.

### 7.8.7. Copy_Sub_Array

Arguments: a character string;
Sub-commands: "Onto_Itself" or a character string;
Help strings: "Copy part of a variable within an existing data structure.", "The name of the variable to be worked on." and "The name of the variable for the results.".

When using this command the user is prompted for information on which parts of the source array to copy and which parts of the destination array to over-write. These are specified in terms of an increment and offset pair for each array subscript in each case. Thus it is possible to copy rows into columns, use every second element, etc.

### 7.8.8. Sum_Values

Arguments: a character string;
Sub-commands: a character string;
Help strings: "Add two variables within an existing data structure.", "The name of the first variable to be used." and "The name of the second variable to be used.".

### 7.8.9. Subtract_Values

Arguments: a character string;
Sub-commands: a character string;
Help strings: "Subtract two variables within an existing data structure.", "The name of the first variable to be used." and "The name of the second variable to be used.".

### 7.8.10. Multiply_Values

Arguments: a character string;
Sub-commands: a character string;
Help strings: "Multiply two variables within an existing data structure.", "The name of the second variable to be used." and "The name of the first variable to be used.".

### 7.8.11. Divide_Values

Arguments: a character string;
Sub-commands: a character string;
Help strings: "Divide two variables within an existing data structure.", "The name of the first variable to be used." and "The name of the second variable to be used.".

### 7.8.12. User_Defined_Function

Arguments: a character string;
Sub-commands: a character string;
Help strings: "Combine two variables within an existing data structure.", "The name of the first variable to be used." and "The name of the second variable to be used.".

This command invokes the routine "USRDF2" to manipulate the variables. This double precision function takes two double precision arguments and must be supplied and linked in by the user when the package is compiled. This feature enables the user to provide a problem specific manipulation if desired. It is invoked by a nested ‘DO’ loop that applies the function systematically to all members of the variables passed to it (n.b., if the two variables have different sizes the one with fewer elements restricts the range of application). If no user defined function is provided a default is used which just sets the result to zero.

### 7.8.13. Main_System

Arguments: none;
Sub-commands: none;
Help strings: "Return to the main system or program.".

This command returns control to the routine that called the transformation facilities, usually the previous menu level.

# CHAPTER 8

# Miscellaneous Topics

## 8.1. Introduction

This chapter details a number of the more advanced features that can be used to gain the most from the **OPTIONS** package together with various other miscellaneous facilities.

## 8.2. Monitoring Execution Status

When carrying out long optimization runs it is often desirable to monitor the progress of the method being used and, sometimes, to be able to save intermediate results. This is achieved using the third argument of the optimization commands in the main optimization menu. This argument should be an integer and represents the number of design evaluations that should be carried out between outputs. If the number is given as negative the monitoring routine waits after each sequence of calculations allowing the user to examine or save the results achieved thus far. The user should be aware, however, that unless the increments in monitoring are carefully chosen (such as to be at the end of individual generations of a GA search) the results shown may not be the 'best so far'. Using negative increments permits backups of long jobs to be produced if required. It also allows the experienced user to control some of the parameters being used during the search. However, under no circumstances should the dimensions of data-base variables be altered, elements added or deleted during runs as this will result in unpredictable results. The user may, however, modify the contents of individual data-base elements if desired. Even so, if these are controlling the optimization process the user must be careful to check for any unexpected effects.

## 8.3. Complex Design Codes

It is often useful when dealing with complex problems to set up general purpose versions of the routines OPTFUN and OPTCON. These can then be used for examining a range of related problems without the need to recode and recompile the resulting programs. This approach suffers from one main disadvantage: if the functions being dealt with are complex they may be time consuming to execute and, if general purpose routines are produced, some of this effort will be wasted if not all of the code is required to calculate the currently selected objective function and constraints. To overcome this difficulty a routine called OPTCHK is provided within the **OPTIONS** package. This routine is then inserted at strategic points throughout the user's code and it is then capable of telling the user supplied code that sufficient work has been done and the objective function or constraints placed in the design data-base. The user produced routines may then return without wasting further time. Appendix J contains an example version of OPTFUN (containing a separate entry point for OPTCON) which uses this mechanism. Here, the problem has been broken into a number of separate design calculations which follow naturally from each other. After each stage of the design, checks are made and the routine returns if sufficient has been done. By using one routine with two entry points the same structure is readily applied to both the objective and the constraints. This is a quite general approach and allows the design code to be very flexible in use, without the need to recode and recompile.

## 8.4. Objective Function Gradient Calculations

Some optimization methods are able to make use of the gradient of the objective function being optimized if this information is known. When it is not, such methods usually synthesize the gradients numerically by finite differencing; this is the approach use by all such methods within the **OPTIONS** package. Sometimes, however, the user is able to calculate objective function gradients directly, and with less effort than the further objective function evaluation in each design direction require by finite differencing. In such circumstances a facility within **OPTIONS** may be used to take advantage of such calculations. To enable this facility, when gradients are being calculated, the GENDAT variable OPT_GRADNT is used, if it exists, to indicate the direction from, offset from and function value at the base point being used. The data available within this variable, together with the results from direct gradient calculations may then be used with the user supplied routine OPTFUN as follows:

> Before running the optimization process, OPT_GRADNT should be defined with arrays subscripts of 1, 1 and 4. Then, when a gradient calculation is being carried out the direction is given in the first and second elements of the array (with the record number of the variable being changed given as the first, see GETVN on how to turn this into a record name and the stack pointer as the second, see GSTACK on how to get the value of the variable and GETVI on how to see which element is being used if the variable is an array). The offset is held as the third element of the array (this is the amount the variable has been incremented from the base point prior to calling the objective function). Finally, the fourth element is used to hold the base point objective function value. To use this information the relevant variable should be decremented by the offset and the gradient calculated at the base-point. This gradient times the offset plus the function value at the base point should then be stored by OPTFUN instead of the objective function value normally used. Then, when the optimizer applies its finite differencing algorithm to the returned data it will accurately recover the desired gradient. Moreover, most such routines calculate the gradient in several directions from a single base-point and the user may repeat this process for all directions requested, often with a single gradient calculation call.

It should be noted that not all routines within the **OPTIONS** package make use of this facility and that many calls to OPTFUN are made when gradients are not desired. Therefore the elements of OPT_GRADNT are all reset to zero when not calculating gradients. OPT_GRADNT(1,1,1) can thus be used as a (non-zero) flag to see if gradients are being calculated. See also the index entry for OPT_GRADNT to see which routines within the **OPTIONS** package use this feature. N.b., since penalty functions distort the surface gradients and minimizing/maximizing the logarithm of the objective function does likewise the variable OPT_GRADNT is only invoked where the preceding rubric will sensibly generate gradients.

### 8.5. User Defined Optimization Routines

Occasionally, a user may have a custom designed optimization routine that is to be used instead of, or alongside, those routines built into or linked with the **OPTIONS** package. To enable such routines to be easily merged into **OPTIONS**, two stub routines are provided in the main library that can be replaced by user defined code. These routines are called OPTUM1 and OPTUM2; the source codes for these routines are supplied as optum1.f and optum2.f. Details of the relevant SUBROUTINE statements required by **OPTIONS** are included in these files to enable the user to correctly interface new versions.

The user supplied routines are accessed from the "Misc. Optimizers Menu" of the main **OPTIONS** menu as the "U1_Optimizer" and the "U2_Optimizer". When used these commands can be supplied with up to two numeric arguments which are then passed to the user's code for whatever action is considered necessary; if no arguments are input by the user these are both set to 0.0 before calling the user's code. The user's code should then *minimize* the current design problem by making calls via the routines DUREAL, DEQUAL and DCONST to the user's problem definition. Again, the sources for the stub routines illustrate the correct implementation of these calls. N.b., by using these standard routines full data-base, graphics and monitoring support are provided to the user supplied methods.

### 8.6. Merging Data-bases

It is sometimes useful to merge two or more data-bases that have been produced for different, but related problems. To enable this process to be carried out, and also to facilitate the production of new data-bases a utility program called "gencmd" is supplied with the **OPTIONS** package. This utility reads a data-base file and translates it into a macro command file which, when executed, recreates the original data-base. If this process is carried out for two or more data-bases the resulting macro files can be readily merged with the system editor. The merged macro file can then be used to create the desired data-base. Alternatively, the macro files may be used as skeletons and edited to design a completely new data-base. When the revised macro file is executed any inconsistencies or duplications are detected by the software and processing of the problem halted at the relevant point. This allows errors in the macro file to be corrected and ensures a valid data-base is produced. The macro files are also more portable than the data-files and may be used for archiving purposes, ensuring that they will be useful without modification in future versions of the package.

## 8.7. Using UNIX System Calls to Link Codes

It is sometimes desirable, when using existing design codes to form the user supplied routines, to be able to execute these codes without turning them into subroutines that can be called by OPTFUN or OPTCON. This may be achieved using system calls under UNIX which are placed in OPTFUN, etc. and which, in turn, execute the design codes. The codes must be set up to place their output in files (usually by UNIX redirection using the ">" symbol) which can then be processed to extract the results of the runs. The utility "optopt" supplied with the package uses this approach and its version of OPTFUN held as "optoptfun.f" demonstrates this process.

## 8.8. System Variables

The **OPTIONS** package uses a number of GENDAT system variables to control various defaults and options. By adding these variables to the GENDAT data-base the user may gain control over these facilities. The beginner is well advised not to interfere with them. Their values are given to the user with a warning when the defaults are invoked, this is normal. Some of the variables are generic in action in that they affect more than one aspect of the package or more than one optimization method while some are specific to individual optimizers: they are described in the following sub-sections while Appendix L details which parts of the code refer to which variables and the default values used. Note that the skeleton data-base file "opdata.d", supplied with the package, contains definitions for all of the standard system variables. The macro command file "params.dbs" contains the commands to add all these variables, along with sensible default values, to a data-base if they are required (this will need editing if a subset of the variables already exist in a data-base; note also that the variables OPT_TOL and OPT_STEP have different meanings for different optimizers and therefore no single value will suit all methods for these two variables, they should therefore only be used with extreme caution - "params.dbs" creates these variables but does not set values for them).

### 8.8.1. Generic Variables

OPT_TOL

controls the accuracy with which solutions are found, it varies from method to method, see the details of individual routines in the respective references (XTOL for the routine E04UAF and "Optimality Tolerance" for E04UCF in the NAg manuals, in both cases setting the accuracy of the solution; TOL in the OPTIVAR manual, applying only to penalty functions OPTIM2, OPTIM3 and OPTIM5 where it is the fraction of the last objective function value that is considered an acceptable maximum change between successive runs for a presumed optimum to have been reached; EC, ED, EPSILO(3) and EPSILO(4) in the Schwefel library of routines where these are used to set the accuracy with which the objective function is achieved, i.e., when the function changes by less than EC or EPSILO(3) absolute or ED or EPSILO(4) relative, an optimum is presumed to have been reached, except in routine POWE where 0.01/OPT_TOL is used to set the variable ESCALE and in routine SIMP where OPT_TOL is used to set REQMIN), default varies from 0.001 to 1.0E-5 depending on the routine used (but the value used is always displayed with a warning that the default is in use);

OPT_CTOL

limits the accuracy with which constraints must be met to be considered satisfied, default 0.001;

OPT_STEP

controls the step size used by the various methods, it varies from method to method, see the details of individual routines in the respective references (STEPMX for the routine E04UAF in the NAg manuals, which specifies an estimate of the Euclidean distance between the minimum and the starting point; F, 1/NSMAX, STEPMX or EPS in the OPTIVAR manual where it is used as the fraction of the range of the design variables when computing partial derivatives except for ADRANS where it is one over the maximum number of consecutive random points generated without an improvement (and so is limited to be less than 0.1), PDS where it is simply the upper limit on the step size taken as a fraction of the variable ranges, RANDOM where it is one over the maximum number of times constraints may be consecutively violated before abandoning the search (and so is again limited to be less than 0.1) and REDUCE where it is the increment used in calculating numerical derivatives; EA, EB, EPSILO(1) and EPSILO(2) in the Schwefel library of routines where these are used to set lower limits on step sizes or required accuracy of solution in the design variables, except in routine COMP where EA is the distance variables are reset when they violate constraints and in routine POWE where OPT_STEP is used to set an array of required relative accuracies for each design variable, EA and EPSILO(1) being absolute values and EB and EPSILO(2) being relative versions scaled by the variable ranges supplied), default varies from 0.1 to 1.0E-6 depending on the routine used (but the value used is always displayed with a warning that the default is in use), this parameter is also used when plotting objective functions where a default of 0.1 is taken to indicate that steps of one tenth of the variable ranges should be used;

OPT_NCALLS
: is loaded with the current value of NCALLS which records how many calls have been made to the user's objective function since the start of this optimization run (this allows user provided code to take any actions needed as a search progress, e.g., a change in accuracy in the final stages of a search);

OPT_TRACE
: is filled with the latest optimization path if it is available, to allow subsequent plotting on contour maps, etc., see the macro file "opt_trace.gr" and graphics style file "opt_trace" (also holds the objective function values at the points used);

OPT_NCCALL
: is loaded with the number of calls that have been made to the user's constraints since the start of this optimization run (this allows user provided code to take any actions needed as a search progress, e.g., a change in accuracy in the final stages of a search);

OPT_CTRACE
: is filled with the latest optimization path and constraint information at each point if it is available. Each constraint is stored in a separate X value from subscript values of 2 to 99 (subscript 1 is reserved for the variable values). There are then up to 9999 data points in the Z slots. The Y subscript spans the constraint value first followed by the equality or lower limit value (for equality, lower limit inequality and range constraints) then the upper limit values (for upper inequality and range constraints). This allows for problems with up to 99 variables and 98 constraints.

OPT_TSIZE
: if defined this parameter fixes the size of the Z subscript in the arrays OPT_TRACE and OPT_CTRACE when tracing is being used; this allows the user to choose the most efficient layout of the OPT_TRACE and OPT_CTRACE arrays to suit the optimization runs being used (ideally, it should be set equal to the number of optimization steps being used if this is below 10,000, or to a simple fraction of this size if longer runs are to be used - n.b., for OPT_TRACE the X subscript is for variablse and objective function, while Y and Z are for evaluation numbers); default 1,000;

OPT_RESLTS
: if defined this parameter is searched for previous results before calling the user supplied objective function routine during optimization runs, its format is the same as for OPT_TRACE so that if OPT_TRACE is renamed to OPT_RESLTS (or the results from OPT_TRACE are copied to OPT_TRACE using the "Copy_Sub_Array" option of the transform data menu) subsequent runs of an optimizer will not need to calculate objective functions (which can be useful when a single such calculation takes a significant time - DO NOT place data in just this array if constraints are also time consuming to compute, use OPT_CRSLTS as well);

OPT_CRSLTS
: if defined this parameter is searched for previous constraint results before calling the user supplied constraint routine during optimization runs, its format is the same as for OPT_CTRACE so that if OPT_CTRACE is renamed to OPT_CRSLTS (or the results from OPT_CTRACE are copied to OPT_CTRACE using the "Copy_Sub_Array" option of the transform data menu) subsequent runs of an optimizer will not need to calculate constraints (which can be useful when a single such calculation takes a significant time - DO NOT place data in just this array if the objective function is also time consuming to compute, use OPT_RESLTS as well);

OPT_GRADNT
: if defined this parameter allows user supplied code to tell when an optimizer is using finite differencing to calculate objective function gradients, see the section in this chapter on "Objective Function Gradient Calculations";

OBJ_THETA
: if defined this parameter is used to hold the $\theta$ parameters in the random process model of the objective function when carrying out Response Surface Modelling (this is an array that will be sized and filled when the model is created and can be amended by hand if desired - it should be cleared to force a new set of parameters to be calculated), see the chapter on Response Surface Modelling for full details;

L_OBJ_THET
: if defined this parameter is used to hold a lower limit value for OBJ_THETA which can be used when carrying out advanced work on response surface models;

U_OBJ_THET
: if defined this parameter is used to hold an upper limit value for OBJ_THETA which can be used when carrying out advanced work on response surface models;

OBJ_EXP
: if defined this parameter is used to hold the $p$ exponent parameters in the random process model of the objective function when carrying out Response Surface Modelling (this is an array that will be sized and filled when the model is created and can be amended by hand if desired - it should be cleared to force a new set of parameters to be calculated), see the chapter on Response Surface Modelling for full details;

L_OBJ_EXP
: if defined this parameter is used to hold a lower limit value for OBJ_EXP which can be used when carrying out advanced work on response surface models;

U_OBJ_EXP
: if defined this parameter is used to hold an upper limit value for OBJ_EXP which can be used when carrying out advanced work on response surface models;

OBJ_LAMBDA
: if defined this parameter is used to hold the regularization parameter in the random process model of the objective function when carrying out Response Surface Modelling, see the chapter on Response Surface Modelling for full details;

L_OBJ_LAMB
: if defined this parameter is used to hold a lower limit value for OBJ_LAMBDA which can be used when carrying out advanced work on response surface models;

U_OBJ_LAMB
: if defined this parameter is used to hold an upper limit value for OBJ_LAMBDA which can be used when carrying out advanced work on response surface models;

OBJ_CLF
: if defined this parameter is used to hold the logarithm of the concentrated likelihood function for the random process model of the objective function when carrying out Response Surface Modelling (this is filled when the model is created), see the chapter on Response Surface Modelling for full details;

CST_THETA
: if defined this parameter is used to hold the $\theta$ parameters in the random process model of the first constraint when carrying out Response Surface Modelling (this is an array that will be sized and filled when the model is created and can be amended by hand if desired - it should be cleared to force a new set of parameters to be calculated), see the chapter on Response Surface Modelling for full details;

L_CST_THET
: if defined this parameter is used to hold a lower limit value for CST_THETA which can be used when carrying out advanced work on response surface models;

U_CST_THET
: if defined this parameter is used to hold an upper limit value for CST_THETA which can be used when carrying out advanced work on response surface models;

CST_EXP

if defined this parameter is used to hold the $p$ exponent parameters in the random process model of the first constraint when carrying out Response Surface Modelling (this is an array that will be sized and filled when the model is created and can be amended by hand if desired - it should be cleared to force a new set of parameters to be calculated), see the chapter on Response Surface Modelling for full details;

L_CST_EXP

if defined this parameter is used to hold a lower limit value for CST_EXP which can be used when carrying out advanced work on response surface models;

U_CST_EXP

if defined this parameter is used to hold an upper limit value for CST_EXP which can be used when carrying out advanced work on response surface models;

CST_LAMBDA

if defined this parameter is used to hold the regularization parameter in the random process model of the first constraint when carrying out Response Surface Modelling, see the chapter on Response Surface Modelling for full details;

L_CST_LAMB

if defined this parameter is used to hold a lower limit value for CST_LAMBDA which can be used when carrying out advanced work on response surface models;

U_CST_LAMB

if defined this parameter is used to hold an upper limit value for CST_LAMBDA which can be used when carrying out advanced work on response surface models;

CST_CLF

if defined this parameter is used to hold the logarithm of the concentrated likelihood function for the random process model of the first constraint when carrying out Response Surface Modelling (this is filled when the model is created), see the chapter on Response Surface Modelling for full details;

RSM_NSKIP

if defined this parameter is used to set the number of basis functions to skip when constructing a regression radial basis function model of the objective function for Response Surface Modelling, see the chapter on Response Surface Modelling for full details;

RSM_NCSKIP

if defined this parameter is used to set the number of basis functions to skip when constructing regression radial basis function models of the constraints for Response Surface Modelling, see the chapter on Response Surface Modelling for full details;

RSM_NULLH

if defined this parameter is used to record the null hypothesis values found when constructing a regression radial basis function model of the objective function for Response Surface Modelling (this is filled when the model is created but must be sized before hand), see the chapter on Response Surface Modelling for full details;

RSM_CNULLH

if defined this parameter is used to record the null hypothesis values found when constructing regression radial basis function models of the constraints for Response Surface Modelling (this is filled when the model is created but must be sized before hand), see the chapter on Response Surface Modelling for full details;

RSM_NULL_T

if defined this parameter sets the null hypothesis threshold (as a %) used in constructing the regression radial basis function models of the objective function or constraints when carrying out Response Surface Modelling, see the chapter on Response Surface Modelling for full details;

RSM_SVDTOL

if defined this parameter is used to hold the singular value tolerance used when applying SVD equation solutions in Response Surface Modelling; see the chapter on Response Surface Modelling for full details;

RSM_EIF_WT
: if defined this parameter is used to hold the weighting between exploitation and exploration used when applying expected improvement methods in Response Surface Modelling; see the chapter on Response Surface Modelling for full details;

RSM_QCK_HP
: if defined and set equal to the number of design variables in a problem this parameter is used to turn on quick SPM hyper-parameter tuning by ensuring that the theta and exponent paramters are all set equal to each other; see the chapter on Response Surface Modelling for full details;

WORK
: is used for holding data when preparing plots of the optimization surface (attempting to make such plots without this variable being present results in an error) or any other activities requiring scratch space (this variable is so commonly used that it is not included in the macro file "params.dbs").

### 8.8.2. NAg Variables

NAG_BIGBND
: is a parameter to the NAg routine E04UCF and is used to set the size of non-existent upper bounds, beyond which they are assumed to lie at $\infty$, default 1.0E10;

NAG_ETA
: is a parameter to the NAg routine E04UAF and is used to set the accuracy of the linear minimizations used by the method, default 0.5;

NAG_RHO
: is a parameter to the NAg routine E04UAF and is used in the definition of the augmented Lagrangian function, default 1.0;

### 8.8.3. Bit Climbing Variables

BC_NBIN
: the algorithm works with a binary discretization; this number controls the number of bits used per variable from two up to a maximum, currently of 16, set internally within the routine, default 12;

BC_PENAL
: this parameter is used to set the penalty function control parameter, $r$, with values less than one invoking the modified Fiacco and McCormick function (OPTIM2) otherwise the one pass method is used (OPTIM1), default 1.0E20, maximum 1.0E30;

BC_NRANDM
: the random number sequence used by the optimizer can be altered by setting this variable which fixes the number of random numbers drawn of and discarded before starting the process, default zero.

### 8.8.4. Dynamic Hill Climbing Variables

DHC_INITSZ
: the algorithm works by carrying out hill climbs looking for local maxima and then restarting at points well away from the maxima already found; this parameter is used to set the non-dimensional size of the initial steps in the hill climbing search and thus must lie between zero and one, default 0.5;

DHC_THRESH
: the hill climbing searches proceed with reducing step sizes until they are less than the value set by this parameter, which must thus also lie between zero and one, default 0.01;

DHC_PENAL
: this parameter is used to set the penalty function control parameter, $r$, with values less than one invoking the modified Fiacco and McCormick function (OPTIM2) otherwise the one pass method is used (OPTIM1), default 1.0E20, maximum 1.0E30;

DHC_NRANDM
: the random number sequence used by the optimizer can be altered by setting this variable which fixes the number of random numbers drawn of and discarded before starting the process, default

zero.

**8.8.5. Population Based Incremental Learning Variables** The PBIL algorithm works by maintaining a probability vector which controls the generation of new trial points at a bitwise level. As the solution progresses this vector, which starts with all elements at 0.5, is evolved to form an accurate template for high quality solutions, i.e., whether bits in the design vectors should, on balance, be zero or one. This process is governed by a learning rate and a mutation probability. These are applied to the probability vector each generation using the best solution of the generation and, when an elite solution appears, within a generation.

PL_NBIN
: the algorithm works with a binary discretization; this number controls the number of bits used per variable from two up to a maximum, currently of 16, set internally within the routine, default 12;

PL_NPOP
: the algorithm works by forming a given number of random guesses and then using the best of these to update the probability vector; this number is called the population and is set by this variable (which must be greater than two), default 100;

PL_PENAL
: this parameter is used to set the penalty function control parameter, $r$, with values less than one invoking the modified Fiacco and McCormick function (OPTIM2) otherwise the one pass method is used (OPTIM1), default 1.0E20, maximum 1.0E30;

PL_LRATE
: the learning rate controls how rapidly the probability vector changes towards the successful solutions at the end of each generation (or when an elite solution is found); this number sets the rate, default 0.05, min. = 0.0, max = 1.0;

PL_PMUTNT
: mutation is applied to the probability vector randomly at the end of each generation, this parameter governs the probability with which any element of the vector is mutated, default 0.02, min. = 0.0, max = 1.0;

PL_NRANDM
: the random number sequence used by the optimizer can be altered by setting this variable which fixes the number of random numbers drawn of and discarded before starting the process, default zero and

PL_PVECTR
: a variable used to store the final PBIL probability vector values and then to use them when restarting the PBIL, thus allowing a PBIL run to be broken into sequences or continued beyond the anticipated completion, no default value and must be sized before calling the PBIL by hand to at least one by the number of variables by PL_NPOP.

**8.8.6. Numerical Recipes Variables**

Four of the methods described in the Numerical Recipes cook-book are avialable along with two alternative and faster conjugate gradient searches.

NRP_DX
: the length of steps taken when using finite differencing to calculate local gradients, default 1.0E-7.

**8.8.7. Design of Experiment Variables**

Seven Design of Experiment search methods are available: the first simply uses random numbers to chose points to sample and returns the best found while the second uses $LP_\tau$ sequences (currently allows up to 51 variables and 16,000 different evaluations)[11], the third uses a central composite design augmented by $LP_\tau$ sequences, the fourth uses a face centred cubic full factorial design augmented by $LP_\tau$ sequences, the fifth and sixth use Latin hypercubes (the sixth is cell based), and the seventh uses a user supplied list of candidate points taken from the DOE_TRACE array (and modified to exclude infeasible points if such information is available in the DOE_CTRACE array - these two arrays should be set up and sized as per the arrays OPT_TRACE and OPT_CTRACE);

DOE_NRANDM

The random number sequence used in the first, fifth and sixth cases can be altered by setting this variable which fixes the number of random numbers drawn of and discarded before starting the process, in the second case this variable is used to alter the point at which sampling of the $LP_\tau$ sequence begins, while in the third and fourth it alters the order in which the design is tested and the $LP_\tau$ elements are added. For candidate list based optimizations the random number sequence alters attempts to tune the list when more candidates are available than needed for the optimization. In any case low prime numbers should ensure different searches for each value used, default zero.

DOE_TRACE

If a user supplied list of test points is to be used for a DoE then this array should contain these points - it is set up exactly as per OPT_TRACE and thus existing trace arrays can be renamed and used for this purpose.

The DOE_TRACE variable can be filled automatically when running GA or DHC based searches. For the GA the returned points are the centroids of the clusters found in the final generation, in arbitrary order. For the DHC the points are the peaks of the function located by the sequence of hill-climb searches, in rank order and excluding that which contains the best value of all (this is returned as the result of the search of course). The number of such points returned is as many as possible limited by the size the user sets on the DOE_TRACE array before the searches are started. The actually number returned varies with the degree of multi-modal behaviour - the bumpier the function the more clusters(GA) / peaks(DHC) that can be located and thus the more points returned. Also a longer DHC search gives more chances to find suitable locations as does a search with a bigger GA population, but this varies widely dependant on many issues and must be investigated by experiment. The user can only check how many points have been returned by looking from the bottom of the DOE_TRACE array up until the first non-zero vector is detected. The screen output, of course shows how many peaks / clusters have been found in total irrespective of the size of the array. The ability to create arrays of points like this can be very valuable when seeking to update a response surface based model, and since DoE searches can be dealt with in parallel such updates points can also be evaluated in parallel.

DOE_CTRACE

If a user supplied list of test points is to be used for a DoE then this array should can contain constraint information such that any points that have violated constrains will be rejected from the trial - it is set up exactly as per OPT_CTRACE and thus existing constraint trace arrays can be renamed and used for this purpose. This enables pre-defined experiments to be rapidly adapted to allow for infeasible regions.

### 8.8.8. Schwefel Library Variables

These variables allow parameters in the evolution strategies supplied by Schwefel to be modified[3]. Their meanings are described in detail in the reference and are only briefly explained here:

SC_LS is a parameter to the Schwefel two-membered evolution strategy EVOL and controls how severe convergence testing is, with bigger values requiring the objective function to remain essentially stationary for longer before convergence is considered complete, default 2;

SC_IELTER

is a parameter to the Schwefel multi-membered evolution strategy KORR and controls the number of parents in a generation, default 10;

SC_NACHKO

is a parameter to the Schwefel multi-membered evolution strategy KORR and controls the number of descendants of a generation, default 100;

SC_NS

is a parameter to the Schwefel multi-membered evolution strategy KORR and controls the number of different step size parameters, default is the number of design variables, $N$;

SC_DELS

is a parameter to the Schwefel multi-membered evolution strategy KORR and controls the global random step sizes, default $1/\sqrt{2N}$;

SC_DELI

is a parameter to the Schwefel multi-membered evolution strategy KORR and controls the local random step sizes, default $1/\sqrt{2N/\sqrt{NS}}$;

SC_DELP

is a parameter to the Schwefel multi-membered evolution strategy KORR and controls the correlation ellipsoid angles, default $5 \times 0.01745 = 5°$;

SC_BKORRL

is a parameter to the Schwefel multi-membered evolution strategy KORR and switches on the rotation of the correlation ellipsoid if non-zero, default 1;

SC_KONVKR

is a parameter to the Schwefel multi-membered evolution strategy KORR and controls how many generations are used when applying convergence tests; default 1;

SC_NRANDM

the random number sequence used by the evolution strategy optimizers EVOL and KORR in the Schwefel package can be altered by setting this variable which fixes the number of random numbers drawn of and discarded before starting the process, default zero.

### 8.8.9. Genetic Algorithm Variables

GA_NBIN

the algorithm works with a binary discretization; this number controls the number of bits used per variable from two up to a maximum, currently of 16, set internally within the routine, default 12;

GA_NPOP

the algorithm works by forming a given number of random guesses and then attempting to improve on them, maintaining a number of best guesses as the process continues; this number is called the population and is set by this variable from two up to a maximum, currently of 500, set internally within the routine, default 50;

GA_PENAL

this parameter is used to set the penalty function control parameter, $r$, with values less than one invoking the modified Fiacco and McCormick function (OPTIM2) otherwise the one pass method is used (OPTIM1), default 1.0E20, maximum 1.0E30;

GA_PBEST

reproduction is related to the objective function so that successful ideas reproduce more often and so are progressed; this parameters sets the proportion of the solutions that are used to form the parents of the next generation, the very best is kept unchanged at each pass for comparison and to ensure that only improvements are allowed (elitism), default 0.8, min. = 0.0, max = 1.0;

GA_PCROSS

this parameter sets the proportion of the solutions in the population that are crossed to form new solutions, this is the key process by which new ideas are formed, based on the successful previous attempts; default 0.8, min. = 0.0, max = 1.0;

GA_PINVRT

this parameter sets the proportion of the solutions in the population that have their ordering codes inverted to form new solutions, the ordering codes allow ideas to be represented by separated binary digits and for these to be moved during crosses *en block*, default 0.2, min. = 0.0, max = 1.0;

GA_PMUTNT

mutation is allowed at a level set by this parameter, i.e., this fraction of the total number of binary digits are reversed at each pass; there is little point in setting this greater than 0.5 since then on average half of each word is reversed and fully random solutions result; some randomisation is required to ensure that all possible options can be reached (at least in theory) and although set at a low level default is vital (setting GA_PBEST to zero and GA_PMUTNT to 0.5 turns this GA into a simple random search, setting GA_PMUTNT negative causes the mutations to be made to the actual variables rather than their binary encodings), default 0.005, min. = 0.0, max = 1.0;

GA_PRPTNL

if this variable is outside the range -0.5 to 0.5 it is taken to be .TRUE. and the make-up of the following generation is then biased in favour of the most successful according to their objective function values, otherwise survival is proportional to ranking but scaled to prevent dominance and stagnation, default 1.0 (i.e., .TRUE.);

GA_ALPHA

this parameter controls the cluster penalising function with small values giving less severe penalties than those nearer one and a value less than zero turning the mechanism off (when it is known that the objective function has many peaks with similar values that can cause premature convergence of the search, this value should be increased from its default, conversely, when there is only one dominant solution and final convergence is a problem it should be set to zero or negative to turn the clustering mechanism off), default 0.2, max. = 2.0;

GA_DMIN

this parameter sets the minimum distance between cluster centroids, as a fraction of the Euclidean distance between opposite corners of the hyper-cube being searched, default 0.05, min. = 0.0, max. = 1.0;

GA_DMAX

this parameter sets the furthest distance a new solution can be from an existing cluster centroid without a new cluster being formed, as a fraction of the Euclidean distance between opposite corners of the hyper-cube being searched, this must be greater than GA_DMIN, default 0.2, min. = GA_DMIN, max. = 1.0;

GA_NCLUST

this parameter sets the initial number of clusters, either in absolute terms or, if it is $\leq 1.0$, as a fraction of the population size, and must therefore lie between zero and one or two and the population size, default 0.2, min. = 2 / GA_NPOP, max. = GA_NPOP;

GA_NBREED

having clustered the current population, breeding is restricted to be between members of the same cluster if there are at least this many members in the cluster, otherwise random members of the population are used, if this parameter is set to zero breeding is not restricted, producing a greater variety of solutions but more "lethals"; if the number is less than unity it is taken to be a fraction of the population, default 0.1, min. = 2 / GA_NPOP, max. = GA_NPOP;

GA_PSEED

seeding of the initial, randomly generated members of the population is allowed at a level set by this parameter, i.e., this fraction of the total number of binary digits in the random population are copies of those in the encoding of the start point of the search; thus a value of zero means these members are truly random, one implies they are simple clones of the start point and values near one that they are strongly biased to lie near the start point in encoding space, default 0.0, min. = 0.0, max = 1.0;

GA_NRANDM

the random number sequence used by the optimizer can be altered by setting this variable which fixes the number of random numbers drawn of and discarded before starting the process (n.b., if GA_POP and GA_CODE are in use, see below, this parameter is updated if it exists at the end of the GA run to reflect the number of random numbers used), default zero;

GA_POP

a variable used to store the GA population values and then to use them when restarting the GA, thus allowing a GA run to be broken into sequences or continued beyond the anticipated completion (it is updated after each population is created and before evaluation so that the user can track populations and make choices depending on population data), no default value and must be sized before calling the GA by hand to at least one by the number of variables by GA_NPOP (if one by the number of variables plus one by GA_NPOP then the additional space is used to store the penalized objective function values as well) and

GA_CODE

a variable used to store the final GA code string values and then to use them when restarting the GA, thus allowing a GA run to be broken into sequences or continued beyond the anticipated completion, no default value and must be sized before calling the GA by hand to at least sixteen by the number of variables by GA_NPOP.

### 8.8.10. Simulated Annealing Variables

SA_NBIN
: the algorithm works with a binary discretization; this number controls the number of bits used per variable from two up to a maximum, currently of 16, set internally within the routine, default 12;

SA_PTEMP
: this parameter sets the power to which the number of iterations must be raised to calculate the number of annealing temperatures, with values near one giving many temperatures, it must lie between zero and one, default 0.3333;

SA_PWIDTH
: this parameter controls the range of temperatures in the annealing schedule, with large values giving a wide range of temperatures, which carries the risk of rapid freezing but gives a wider ranging search (n.b., a width of one means all trials are carried out at a single temperature), it must lie in the range 1-10, default five;

SA_PCOLD
: this parameter controls the bottom temperature in the annealing schedule, with values over two giving lower temperatures and thus more accurate results at the expense of perhaps missing the global optimum, it must lie in the range 0.1-10, default two;

SA_SCHED
: if this parameter exists and contains an array of variables it is taken to be a cooling schedule which is to be used in place of the preceding three parameters, n.b., only the Z part of the array is used and its length fixes the number of cooling temperatures to be used (it is best to examine a trial run of the SA without using this array before choosing suitable values), default undefined;

SA_PENAL
: this parameter is used to set the penalty function control parameter, $r$, with values less than one invoking the modified Fiacco and McCormick function (OPTIM2) otherwise the one pass method is used (OPTIM1), default 1.0E20, maximum 1.0E30;

SA_PMUTNT
: mutation is allowed at a level set by this parameter, i.e., this fraction of the total number of binary digits are reversed at each evaluation (setting SA_PMUTNT negative causes the mutations to be made to the actual variables rather than the binary digits), default 0.1 and

SA_NRANDM
: the random number sequence used by the optimizer can be altered by setting this variable which fixes the number of random numbers drawn of and discarded before starting the process, default zero.

### 8.8.11. Evolutionary Programming Variables

EP_NBIN
: the algorithm works with a binary discretization; this number controls the number of bits used per variable from two up to a maximum, currently of 16, set internally within the routine, default 12;

EP_NPOP
: the algorithm works by forming a given number of random guesses and then attempting to improve on them, maintaining a number of best guesses as the process continues; this number is called the population and is set by this variable from two up to a maximum, currently of 500, default 50;

EP_PENAL
: this parameter is used to set the penalty function control parameter, $r$, with values less than one invoking the modified Fiacco and McCormick function (OPTIM2) otherwise the one pass method is used (OPTIM1), default 1.0E20, maximum 1.0E30;

EP_IMUTNT
: mutation is controlled so that the best members are mutated least and the worst, most, this parameter governs the order of the mutation with ranking, a value of one thus gives a linear change, two a quadratic one and so on (only positive values being allowed), default two;

EP_TOURN

this parameter sets the number of members in the ranking tournament, either in absolute terms or, if it is ≤ 1. 0, as a fraction of the population size, and must therefore lie between zero and one or two and the population size, default 0.5, min. = 2 / EP_NPOP, max. = EP_NPOP and

EP_NRANDM

the random number sequence used by the optimizer can be altered by setting this variable which fixes the number of random numbers drawn of and discarded before starting the process, default zero and

EP_POP

a variable used to store the final EP population values and then to use them when restarting the EP, thus allowing an EP run to be broken into sequences or continued beyond the anticipated completion, no default value and must be sized before calling the EP by hand to at least one by the number of variables by EP_NPOP.

### 8.8.12. Evolutionary Strategy Variables

ES_NPPOP

the algorithm works by forming a given number of random guesses called parents, it then breeds the parents to produce children which, after directed mutation, hopefully contain better solutions, the children and parents being then merged to form new parents before the process is repeated; these numbers are called the populations, this parameter sets the parent population size from two up to a maximum, currently of 500, default 100;

ES_NCPOP

the algorithm works by forming a given number of random guesses called parents, it then breeds the parents to produce children which, after directed mutation, hopefully contain better solutions, the children and parents being then merged to form new parents before the process is repeated; this parameter sets the parent population size, either in absolute terms or, if it is ≤ 1. 0, as a fraction of the population size, and must therefore lie between zero and one or two and the population size, default 1.0, min. = 2 / ES_NPPOP, max. = 500;

ES_PENAL

this parameter is used to set the penalty function control parameter, $r$, with values less than one invoking the modified Fiacco and McCormick function (OPTIM2) otherwise the one pass method is used (OPTIM1), default 1.0E20, maximum 1.0E30;

ES_DELSIG

mutation is controlled by dynamically adapted variables whose changes at any one generation are controlled by this parameter; it is used to set the standard deviation of a random number whose exponential is then used to scale the previous mutation control parameter, a small value thus means small changes are made in the search to begin with while a large value gives dramatic changes; it is restricted to lie between zero and 10.0, default 0.1;

ES_UCHILD

when selecting the next generation all the children may be used or a mixture of the best children and parents used; if this parameter is non-zero it is taken to be .TRUE. and the children are used in preference to parents, default zero (.FALSE.);

ES_VDSCRT

when mating two parents the child may inherit variables from one parent or the other (discrete crossover) or the average of the two (intermediate crossover); this process is applied to both the design variables and the mutation control parameters; this variable controls the design variables; it is taken to be .TRUE. if non-zero and discrete cross-over is then used, default one (.TRUE.);

ES_MDSCRT

when mating two parents the child may inherit variables from one parent or the other (discrete crossover) or the average of the two (intermediate crossover); this process is applied to both the design variables and the mutation control parameters; this variable controls the mutation control parameters; it is taken to be .TRUE. if non-zero and discrete cross-over is then used, default zero (.FALSE.);

ES_NRANDM

the random number sequence used by the optimizer can be altered by setting this variable which fixes the number of random numbers drawn of and discarded before starting the process, default zero and

ES_PPOP

a variable used to store the final ES parent population values and then to use them when restarting the ES, thus allowing an ES run to be broken into sequences or continued beyond the anticipated completion, no default value and must be sized before calling the ES by hand to at least one by the number of variables by ES_NPPOP.

### 8.8.13. Statistical Analysis Variables for Stochastic Methods

It is often useful when running stochastic methods that use sequences of populations to analyse the statistics of individual populations. The genetic algorithm, evolutionary programming, evolution strategy and populations based incremental learning routines all include such a facility, using a number of system variables if these are present. The statistics for each population are kept separately so that these may be plotted subsequently by using the Z subscript of the variables: the range of this subscript must be set manually before applying the methods. The routine GA uses any or all of the following:

MN_GA

for the mean,

SD_GA

for the standard deviation,

MS_GA

for the mean + standard deviation ( - SD if mean < 0),

SK_GA

for the skewness coefficient,

KU_GA

for the kurtosis,

GM_GA

for the Geometric mean,

GSD_GA

for the Geometric standard deviation,

MD_GA

for the Mode, that is, the most probable value, taken to be the centre of the class containing the mode,

P05_GA

for the 5% probability level,

P50_GA

for the Median (50% prob. level),

P95_GA

for the 95% probability level,

MIN_GA

for the Minimum value and

MAX_GA

for the Maximum value.

The routines EP, ES and PBIL use similarly named variables but with "GA" replaced by "EP", "ES" and "PL", respectively. The design of experiment routines also allow for this form of analysis but with "GA" replaced by "DOE". In the case of EP and ES the statistics refer to the parent populations, although if the Y subscript of the variables is set to 2 then the child statistics are also kept (i.e., in elements 1, 2, generation number). The style files "opt_ga", "opt_ep", "opt_es" and "opt_pl" may be used to plot the more important of these quantities against generation number if the latter are stored in the array WORK (usually set-up by selecting the array dimension by hand and inserting a unit increment and start). Note that if the optimizers are trying to maximize the objective function all the statistics will be based on function values multiplied by -1. Also note that the statistics are based on the objective function values including penalties for constraint violations.

### 8.9. Optimizing Optimizer Control Parameters

As has been noted in the previous section, the various optimizers available in the **OPTIONS** package can be controlled by the use of a number of control parameters as well as the normal command arguments. The correct selection of these parameters and arguments, to gain the best performance on any specific problem, is by no means obvious. This raises the potential need to optimize these parameters themselves, although general purpose defaults are supplied. Such optimization is only feasible where individual optimizations runs are relatively rapid but this is sometimes possible, particularly if cut-down versions of the user specific design code are available. To carry out this process the package comes with a ready built utility program called "optopt". The data-files for this program take a highly stylised form where the comment section is used to pass information on the desired **OPTIONS** program to run together with its input data-file. Specifically the data-usage block must contain three lines like:

Problem program name = beam1
Problem data-file name = beam1.d
Problem type = Minimize

These lines define the name of the program to run, its data-file name and the type of search to be carried out (which could also be "Maximize", "LMinimize" or "LMaximize" - the last two options working on logarithms of the objective function). The data-usage block can also contain lines like:

Problem vector = 2/3/4
Problem command = yyy

which allow the user to indicate which element of a vector the objective function is to be (which must only be used if the objective function is not a scalar) and also a command (or commands) which must be executed before starting the optimization process once the relevant data-file has been loaded (or to get to the main optimizer menu if **OPTIONS** is being used as part of a larger package). N.b., if "optopt" is to be used the user *must* run the macro file "params.dbs" on the problem specific code to augment its data-base before "optopt" is run.

The package is supplied with a macro command file "optopt.dbs" which can be used to build data-files suitable for use with optopt. Also supplied is the file "optoptfun.f" which contains the OPTFUN and OPTCON routines used by "optopt". This file can be used to produce modified versions of "optopt" if required (n.b., the file is UNIX specific and so "optopt" can only be used on UNIX platforms in its current form). It is set up in such a way that its objective function always indicates faster, better optimizations by lower values, i.e., it is designed to be minimized, even if the underlying problem is one of maximization. Note that the user's code should not generate output containing the word "ERROR" unless this is meant to indicate a fatal error where the results of the calculation are meaningless, i.e., output such as "ERRORS ARE INDICATED BY ..." or "NO ERRORS FOUND" should be avoided.

When the "optopt.dbs" macro has been run only the data usage block needs correctly defining together with the selection of suitable values for some or all of the variables NUM_ITERS, L_NUM_ITER, U_NUM_ITER, D_NUM_ITER INUM_ITERS, POW_ITERS, NUM_REPTS and OPT_TYPE. These variables have the following uses:

NUM_ITERS
: fixes the number of objective function evaluations (iterations) that are to be used when optimizing the user's problem, at least to begin with (default 500).

L_NUM_ITER
: sets the lower limit on NUM_ITERS when seeking the best set of control parameters to use with the user's problem (default 50).

U_NUM_ITER
: sets the equivalent upper limit on NUM_ITERS (default 2,000).

D_NUM_ITER

sets an array of suggested discrete values for NUM_ITERS so that simple round values are used for this parameter (default 50, 100, 150, 200, 300, 500, 800, 1,000, 1,500, 2,000).

INUM_ITERS

defines the ideal number of iterations to be used: "optopt" rewards methods using fewer steps than this and penalizes those taking more (default 500).

POW_ITERS

controls how strongly "optopt" rewards fast solutions with values more than unity tilting the balance in favour of speed while those less than unity favour accuracy (default 0.15; a value of zero removes speed from consideration by "optopt" while values over ten cause speed of solution to dominate the final results).

NUM_REPTS

controls how many averages to be used when applying stochastic techniques; "optopt" runs such methods several times to gain average results (this variable can be selected in the range 1-50, default 5).

OPT_TYPE

controls which optimizer to apply to the user's problem as follows:

| | |
|---|---|
| 1.1 | for OPTIVAR routine ADRANS |
| 1.2 | for OPTIVAR routine DAVID |
| 1.3 | for OPTIVAR routine FLETCH |
| 1.4 | for OPTIVAR routine JO |
| 1.5 | for OPTIVAR routine PDS |
| 1.6 | for OPTIVAR routine SEEK |
| 1.7 | for OPTIVAR routine SIMPLX |
| 1.8 | for OPTIVAR routine APPROX |
| 1.9 | for OPTIVAR routine RANDOM |
| 2.1 | for user specified routine OPTUM1 |
| 2.2 | for user specified routine OPTUM2 |
| 2.3 | for NAg routine E04UCF |
| 2.4 | for bit climbing |
| 2.5 | for dynamic hill climbing |
| 2.6 | for population based incremental learning |
| 2.7 | for numerical recipes routines |
| 2.8 | for design of experiment based routines |
| 2.9 | for design of experiment based routines (but without function calls) |
| 3.11 | for Schwefel library Fibonacci search |
| 3.12 | for Schwefel library Golden section search |
| 3.13 | for Schwefel library Lagrange interval search |
| 3.2 | for Schwefel library Hooke and Jeeves search |
| 3.3 | for Schwefel library Rosenbrock search |
| 3.41 | for Schwefel library DSCG search |
| 3.42 | for Schwefel library DSCP search |
| 3.5 | for Schwefel library Powell search |
| 3.6 | for Schwefel library DFPS search |
| 3.7 | for Schwefel library Simplex search |
| 3.8 | for Schwefel library Complex search |
| 3.91 | for Schwefel library two-membered evolution strategy |
| 3.92 | for Schwefel library multi-membered evolution strategy |
| 4 | for genetic algorithm search |
| 5 | for simulated annealing |
| 6 | for evolutionary programming |
| 7 | for evolution strategy |

(no default is supplied for this variable and the user must therefore select a suitable value or run a macro that does so).

Clearly, the first four parameters here are closely linked to the user's problem. When a problem has many dimensions or may be evaluated quickly more iterations may be used and so these quantities might need to be increased in size. Conversely, very expensive problems might well preclude so many steps being used. The default values merely represent sensible starting points and even with test problems may lead to "optopt" runs of considerable length.

By way of example, consider the two-dimensional bump problem ("bump.d"). To apply "optopt" to this case the data-base created by "optopt.dbs" should have the following lines set in its data usage section:

Problem program name = bump
Problem data-file name = bump.d
Problem type = Maximize
Problem command = mod 2/4 d
Problem command = 1 1 2
Problem command = 1 1 2
Problem command = 1 1 2
Problem command = in nvars 2.0

These commands not only define the required program and data-file names but also cause the program to automatically set the problem size to be two-dimensional before carrying out the desired optimization. The default values of all the other parameters created by "optopt.dbs" are then suitable to begin using "optopt", except for the choice of OPT_TYPE: a good robust starting point for this kind of work is the GA optimizer, i.e., an OPT_TYPE of 4.

To help use "optopt" two further macro command files are supplied with the package called "optopt.run" and "optopt.eval". "optopt.run" attempts to tune each optimizer in turn using the PBIL optimizer to carry out this process. This can be extremely time consuming and the user is advised to simply use parts of this file to study those methods of most interest. To help in this selection the file "optopt.eval" just runs each optimizer in the package once so that a comparison of the various method using default parameters may be made. Both these files systematically vary OPT_TYPE and set up "optopt" to investigate those data-base variables which are relevant to the current method. They also control an additional set of data-base variables that are used to supply suitable command arguments where the relevant optimizers can accept such input, such as the selection of penalty function type with the OPTIVAR routines. These variables have the following uses:

OVR_MAND
: turns on mandatory design constraints with the OPTIVAR routines (1 for on and 0 for for off), default zero;

OVR_PENAL
: selects the kind of penalty function used by a number of the OPTIVAR routines (1 for one pass external, 2 for Fiacco-McCormick, 3 for Powell and 4 for Schuldt), default 1;

OVR_SEED
: sets the seed for random number sequences in the OPTIVAR routine ADRANS, default 128;

OVR_CONV
: sets the convergence criterion for the OPTIVAR routines DAVID, FLETCH, JO, PDS and SIMPLX, default 1D-4/1D-5;

OVR_STOP
: sets the minimum step length stopping criterion for the OPTIVAR routine SEEK, default 0.01;

OVR_STEP
: sets the fraction of range limiting step lengths for the OPTIVAR routine APPROX, default 0.1;

OVR_SIMP
: sets the maximum number of simplex iterations for the OPTIVAR routine APPROX, default 46.0;

OVR_NPTS
: sets the number of points retained per iteration in the OPTIVAR routine RANDOM, default 5;

OVR_SHRK
: sets the shrinkage factor in the OPTIVAR routine RANDOM, default 4;

MC_MAND

turns on mandatory design constraints with some of the misc. routines; (1 for on and 0 for for off), defualt zero;

MC_P1

sets the first parameter for the user supplied optimizers, no default;

MC_P2

sets the second parameter for the user supplied optimizers, no default;

MC_TYPE

selects the kind of optimizer used by the numerical recipes routines (1 for Powell, 2 for Polak-Ribiere, 3 for Fletcher-Reeves, 4 for Broyden-Fletcher, 5 for Gilbert and Nocedal Polak-Ribiere and 6 for Gilbert and Nocedal Fletcher-Reeves) and the design of experiments routines (1 for pure random, 2 for $LP_\tau$, 3 for central composite, 4 for FCC factorial, 5 for Latin hypercubes, 6 for cell-based Latin hypercubes and 7 for a user supplied candidate list taken from the DOE_TRACE array, modified to exclude infeasible points if such information is available in the DOE_CTRACE array), default 1;

MC_PENAL

selects the kind of penalty function used by the numerical recipes routines (1 for one pass external and 2 for Fiacco-McCormick), default 1;

SC_PENAL

selects the kind of penalty function used by unconstrained search methods in the Schwefel library routines, i.e., all methods except the Rosenbrock and evolution strategy methods (1 for one pass external and 2 for Fiacco-McCormick), default 1;

SC_TYPE

is a parameter to the Schwefel multi-membered evolution strategy KORR and the Schwefel library routine POWE, for KORR it controls whether the "comma" or "plus" version of the code is used while for POWE it selects the default convergence criterion or an alternate criterion (1 for "comma" or "default convergence" and 2 for "plus" or "alternate convergence"), default 1;

SC_CONV

is a parameter to the Schwefel library routine DFPS and defines the expected solution value of the objective function at the optimum, default zero (50% improvement);

SC_NITERS

is a parameter to the Schwefel library routine SIMP and fixes the number of iterations before convergence testing is applied, default zero (the total number of function calls to be used divided by 25 times the number of design variables);

SC_LR

is a parameter to the Schwefel two-membered evolution strategy EVOL and controls step size management, with bigger values giving a slower but more accurate search, default 1;

SC_SN

is a parameter to the Schwefel two-membered evolution strategy EVOL and controls step size adjustment, which can be kept constant using a value of unity, default 0.85;

SC_IREKOM

is a parameter to the Schwefel multi-membered evolution strategy KORR and controls the recombination type; default 333 (n.b., each digit in this variable must lie between 1 and 5, i.e. minimum is 111 and maximum is 555)

Finally, it is worth noting that, if computing resources allow, "optopt" should be used with "optopt.eval" on all new problems so as to systematically assess which methods are most appropriate for the problem in hand. Moreover, if speed is not an issue, POW_ITERS should be set to zero before doing this, so as to gain a direct comparison between all methods, unbiased by speed of solution. The output of such runs can then be analysed to decide on which methods to study or tune further.

### 8.10. OPTCNT Common Block

To enable the user supplied objective function and constraint codes to adapt as the optimization progresses it is possible to monitor a common block containing a number of variables of interest. This block is declared as follows

```
C
C          optimizer counter and control common block
C
         INTEGER OPTDIR,MANDDC,NCALLS,MCALLS,NCCLLS,MCCLLS
         REAL RSMOFL,RSMCFL
         COMMON/OPTCNT/OPTDIR,MANDDC,NCALLS,MCALLS,NCCLLS,MCCLLS
     1   ,RSMOFL,RSMCFL
         SAVE /OPTCNT/
```

Here OPTDIR is -1 for minimization (the default), 1 for maximization, -2 for minimization of the logarithm of the objective function and 2 for maximization of the logarithm. MANDDC is set to one if the design vector limits are to be used as constraints on the problem (since some methods otherwise treat them as advisory only), otherwise it is zero. NCALLS is the number of calls made to the objective function routine since starting this optimization run (it is reset before each run and is zero before any run has been made). MCALLS is the total number of calls to the objective function routine asked for when starting this optimization run (it is reset before each run and is zero before any run has been made). NCCLLS is the number of calls made to the constraint routine since starting this optimization run (it is reset before each run and is zero before any run has been made). Thus monitoring NCALLS, MCALLS and NCCLLS allows routines to know how far through a run they are. MCCLLS is currently unused. RSMOFL contains a copy of the response surface modelling code for the objective function, see the definition of OBJMOD in the section concerning OPTRSS Input Arguments for details of these codes. This allows user supplied objective function or constraint codes to know whether or not any RSM models are active. Finally, RSMCFL contains a copy of the response surface modelling code for the constraints, see the definition of CONMOD in the section concerning OPTRSS Input Arguments for details of these codes. This allows user supplied objective function or constraint codes to know whether or not any RSM models are active.

It is of *very great importance* that no attempt is made to change these variables as such modifications may cause the optimizers to behave in unpredictable ways.

### 8.11. Miscellaneous Common Blocks

The **OPTIONS** package additionally makes use of a number of common blocks to transfer data between its internal modules. Therefore the following common block names should be avoided:

CPRTSW, see Appendix B; CPRTNM, see Appendix B;
IOLUN, see Appendix B; CWSTRG, see Appendix B;
DAT001, see Appendix B;
DAT002, see Appendix B;
OPTCNT, holds the optimizer control structure, see previous section;
OPTMZ1, holds the optimizer main variable structure;
OPTMZ2, holds the optimizer secondary variable structure;
OPTTRC, holds the optimizer plot tracing structure;
F1COM, is used to control the Numerical Recipes optimizers;
OPTSC1, is used to control the Schwefel optimizers;
ADRAND, is used by the OPTIVAR package;
APPROD, is used by the OPTIVAR package;
DAVIDD, is used by the OPTIVAR package;
FLETCD, is used by the OPTIVAR package;
JOD, is used by the OPTIVAR package;
PDSD, is used by the OPTIVAR package;
RANDOD, is used by the OPTIVAR package;
REDUCD, is used by the OPTIVAR package;
SEEKD, is used by the OPTIVAR package;
SMPLXD, is used by the OPTIVAR package;
OPTI, is used by the OPTIVAR package;

EVZ, is used by the Schwefel package;
HSU, is used by the Schwefel package;
PIDATA, is used by the Schwefel package;
GDIN01, is used by the GENDAT package;
GDIN02, is used by the GENDAT package;
OSWTCH, is used by the GENDAT package;
GBLCK1, is used by the GENDAT package graphics subsystem;
GBLCK2, is used by the GENDAT package graphics subsystem.

Appendix N details the names of all routines and common block names found with the **OPTIONS** package. Note, however, that in addition, the Simpleplot graphics system uses a number of COMMON blocks internally, typically with names beginning with the letters "JB".

### 8.12. UNIX Environment Variables

To enable the **OPTIONS** package to be neatly installed on UNIX systems with various file layouts, a number of environment variables must be defined to point to the location of key files. This enables these files to be accessed irrespective of where the compiled programs reside. On non-UNIX systems these key files must be placed in the same directory as the running version of the package. The variables are:

GENDAT_GDATA
: which points to the location of a default data-base file which is read in if no other is supplied (this is not usually invoked by **OPTIONS**);

GENDAT_GSTYLE
: which points to the location of a default graphics style file;

GENDAT_HELP
: which points to the location of the on-line help file and

GENDAT_CODES
: which points to the location of the code file.

In addition to these variables which point to files that must be available, the environment variable GENDAT_STARTUP may be used. If defined this variable should point to the location of a macro command file that is to be run immediately a program is loaded. This file may therefore be used to carry out a series of start-up actions that suit the problem in hand.

If a number of UNIX hosts are connected together on a network or the user's machine has multiple processors it is possible to define a parallel processing environment using the UNIX environment variable

OPTIONS_PARALLEL
: which indicates how many jobs to try and run at the same time and where to put them (up to a maximum of 100). The variable contains the number of jobs to run followed by a colon separated list of host names to use in addition to the current local host, it takes the form "njobs:host1,u/l/n:host2,u/l/n" where "njobs" is the number of jobs to run in parallel (including the master process on the current host that is initiating other jobs), "u" is the maximum number of users a machine can have if it is to be used, "l" is the maximum one minute load average a host can have if it is to be used and "n" is the "nice" level to run jobs at (if two numbers are given "u" is set large, while if only one is given both "u" and "l" are set large). If only "njobs" is specified all jobs are placed on the current host. When a job is submitted to a host the value of the one minute load average for that host is incremented by one in the internal tables maintained by **OPTIONS**, even though the load average takes a while to respond. This allows multiple jobs to be rapidly loaded in a sensible fashion.

Note that if hosts are skipped due to being over-loaded or if "njobs" is larger than the number of hosts in the list then the list is reused from the beginning. Thus "njobs" jobs are always started whatever the loading situation. This process is over-ridden, however, if the last host in the list is skipped as being overloaded (which can only occur if at least "l" is specified for that host). This ensures that an infinite loop of skipping hosts and reusing the list does not occur. It also provides the user with a way of automatically controlling not only which hosts are used but also how many jobs are run in parallel. This feature means that the last host in the list should either be the most robust host available (in which case "u" and "l" should be omitted and if all else fails this host carrys all the required jobs) or a host whose load is allowed to control the total number of jobs to be run (since with at least "1" set this host will prevent the host list being reused under high load conditions).

It is quite normal to use OPTIONS_PARALLEL to indicate multiples of the actual number of processors so as to smooth out loading peaks if maximum performance is required and individual job times vary. Similarly, it is quite normal to repeat host names in the list to achieve maximum performance on a network. It is also sensible to actually set "njobs" to use less machines than the list contains to allow for other users loading the network. It is important to realise, however, that when many jobs are run on a single host each job may consume a significant amount of virtual memory and therefore multiply loaded processors should have ample swap space. If insufficient virtual memory/swap space is available then some jobs will fail to start and be transferred back for internal solution by the master processor, thus slowing the whole process down. Finally, it should be noted that OPTIONS_PARALLEL can give the name of a file in the current directory containing the number of jobs and the list of hosts. This file is periodically re-read during processing so that using such a file the user may alter the spread of load and machines in use *during* an optimization run. This is particularly useful on long jobs.

Note that it is wise *not* to define this variable in user log-in files as many simple jobs do not warrant the overhead of parallel processing and so this feature is usually only employed when a big, time consuming job is to be started. In such cases the variable can be defined by hand at the start.

Note also that parallel processing will only be invoked if the routine "main.c" supplied with **OPTIONS** is used as the main calling program for the optimization (as opposed to calling the optimization suite as a set of subroutines). Moreover, to minimize file I/O users should avoid defining large data structures which are not used during the optimization process itself (but only for pre- or post-processing).

OPTIONS_WAIT_TIME

When parallel processing it is often the case that the various jobs running in parallel do not finish at the same time. In such situations the main process waits until all jobs are complete before moving on. To gaurd against the situation where one of the slave parallel jobs is killed before completing, the main process will, in the end, give up waiting and carry out the required calculations internally. Clearly this is wasteful if a few more seconds of waiting would have been sufficient for the slave job to finish. **OPTIONS** attempts to calculate what is a reasonable period to wait and typically this is set equal to the geometric average of all the jobs run so far, over and above, the time needed to carry out the internal processing. The UNIX environment variable OPTIONS_WAIT_TIME can be used to enforce a minimum waiting time if the user believes that the internal calculation may lead to jobs being abandoned too readily. It is specified as an integer number of seconds to wait over and above the internal job time, i.e., the **OPTIONS** master process will wait at least this long for the slave process to finish after it has completed its own calculation.

OPTIONS_LOG_DIR, OPTIONS_LOG_INTERVAL

It is often useful to keep track of previous runs of **OPTIONS** codes if these are computationally expensive to produce. It is also sometimes useful to be able to keep track of user input to be able to create command file sequences after a run has been completed. To facilitate this **OPTIONS** can maintain an archive system on UNIX machines. The archive file is placed in the directory defined by the environment variable OPTIONS_LOG_DIR if defined, or if not "/var/log/options" (provided these directories exists). To prevent files being created when not needed **OPTIONS** will only create an archive entry if a program has been running for at least the time (in seconds) defined by the environment variable OPTIONS_LOG_INTERVAL (n.b., this is wall clock time not CPU seconds). Attempts are then made at strategic points to add to the archive entry, with at least this elapsed time between updates. The default for this variable is 600 seconds while setting it very large clearly prevents logs being created.

The format of the archive entry is of a master file plus standard **OPTIONS** data-files. The data files are detailed in the master file but share the same file-name root as the master file with .d1, .d2, etc appended. They can be re-read directly into the **OPTIONS** program that created them, provided the long file-name system is used to access them. The master file name is constructed by concatenating the program name creating the log with a date code, a time code and the process identifier. The date code is calculated from $(year - 2000) \times 366 \times 31 + (month - 1) \times 31 + day$ while the time code is the number of seconds since midnight, e.g., a file name of "3d_drag_opt_25_31513_159173" indicates that the entry was created by a program called "3d_drag_opt" on the 25th January 2000 at 8:45:13 running as process number 159173. The master file is designed to be human readable and records sufficient information to reproduce whatever operations the user had carried out at the time the log was created.

### 8.13. Error Codes

Sometimes the **OPTIONS** package will fail giving a numeric error code. The meanings and most likely causes of these codes are detailed in Appendix P.

### 8.14. Test Routines, Macros and Makefile

The **OPTIONS** package is supplied with a number of test routines, macros, graphics style files and a "Makefile" which enables a number of sample problems to be compiled and run. These enable the user to verify that the installation is working correctly and also provide templates from which to develop new, problem specific, files. Included among these files are those needed to run the example problems dealt with in this manual. The macro files allow the test routines to be run without further user input. Appendix N contains a list of all the files making up a standard UNIX installation.

The Makefile provides details of the normal compilation actions required when using the package on UNIX workstations with NAg, Simpleplot and XView. "make beam1" and "make beam2" give "beam1" and "beam2" which run the Nowacki beam problem described in the example chapters. "make beam1s", "make beam1sc", "make beam2s" and "make beam2sc" also make programs that will run the Nowacki beam problem but use the direct subroutine version of **OPTIONS** instead of the interactive version, see Chapter 9 (n.b., the "sc" version are subroutine based and written in unix based C). "make beam3" gives "beam3" which runs a multi-objective Pareto variant of the Nowacki beam problem described in chapter 11. "make nag" gives a program called "nag" which is for the NAg examples whilst "make optivr" gives the OPTIVAR example. "make bump" gives a difficult multi-dimensional multiple peak problem where a *maximum* is sought. "make colville" gives the Colville function which was cited in the GA digest volume 7, number 3 and "make jhrr" gives the John Holland Royal Road problem. Finally "make clean" cleans up the **OPTIONS** directory of all running programs, object and graphics output files.

#### 8.14.1. Nowacki Beam Problem

The Nowacki beam problem[14] is held in the routines "beamfun1.f", "beamcon1.f", "beamfun2.f" and "beamcon2.f" which should be used in conjunction with the macro command files "beam1.dbs", "beam1.run", "beam2.dbs" and "beam2.run" to carry out the beam analysis. The simulated annealing routine will deal with the initial, two dimensional, beam problem but with poor accuracy giving a solution after 1,000 iterations of breadth = 16.429mm and height = 16.379cm and an objective function of 2690.76 $mm^2$ compared to the actual solution of 15.800mm, 15.800cm and 2496.64 $mm^2$ respectively. The evolutionary programming routine will also deal with the beam problem, again with poor accuracy giving a solution after 1,000 iterations of breadth = 16.429mm and height = 15.912cm and an objective function of 2614.17 $mm^2$. The dynamic hill climbing method is better getting to 15.824mm, 15.801cm and 2500.35 $mm^2$ in a similar number of iterations. The evolutionary strategy method is the worst of the stochastic approaches giving 17.480mm, 15.458cm and 2702.15 $mm^2$.

#### 8.14.2. NAg Problems

The examples for the NAg routines E04CGF, JAF, UAF and UCF[15, 5] are available as "nagfun.f" and "nagcon.f" and will yield the results given in the NAg manual if all is working correctly. Double precision is used in these routines since the NAg routines make trial steps that only show up in double precision calculations. To run these checks read in the data-files called "e04cgf.d", "e04jaf.d", "e04uaf.d" and "e04ucf.d", respectively and run a NAg optimization with 500 loops. The answers are:

E04CGF : 0.5000, -1.0000 with an optimum of 0.0000;
E04JAF : 1.0000, -0.0852, 0.4093, 1.0000 with an optimum of 2.4338;
E04UAF : 1.2264, 1.4150, 1.4445, 1.5000, 1.5000 with an optimum of 0.0868;
E04UCF : 0.0609, 0.5976, 1.0000, 0.5976, 0.0609, 0.3438, 0.5000, -0.5000 and
-0.3438 with an optimum of -1.3500 .

It is normal for the routine E04UCF to indicate a failure code of 4 (not enough steps) when solving the E04UAF problem and a code of 6 (too great an accuracy requested) when solving the E04UCF problem: in both cases the results are, in fact, correct. Routines E04UAF, JAF and CGF are currently commented out since they are not compatible with latest mark of the NAg library. They may be simply restored if the mark 12 library is used. E04UCF replaces all of them quite acceptably. When the mark 12 library is used and the routines are reinstated E04UCF is chosen by default in favour of E04UAF since the latter is not recommended by NAg. The routines, E04CGF, JAF and UAF/UCF are for unconstrained, bounds constrained and fully constrained problems, respectively. To make E04CGF, JAF and UAF mutually compatible the routine "e04cgy.f" has a fix to make the specification of FUNCT1 the same in all three routines. The altered version is held in "optnag.f" within the package.

### 8.14.3. OPTIVAR Problem

The OPTIVAR example from appendix A of Siddall's book[2] is held in the routines "optivrfun.f" and "optivrcon.f" which should be used in conjunction with "optivr.d" to carry out the example problem analysis. All the OPTIVAR routines solve this problem with reasonable accuracy giving 3.0000, 3.0000, 1.0000 with an optimum of -17.0000. The fastest routine is APPROX which achieves this accuracy in 20 trials. The genetic algorithm routine will also solve the OPTIVAR example but with rather poor accuracy giving a solution of 2.92307, 3.157509, 1.028083 with an optimum of -16.94119 after 1,000 trials. It does slightly better if the clustering algorithm is disabled by setting GA_ALPHA to zero or less, giving a solution of 3.001221, 3.125763, 1.018315 and an optimum of -16.96587. Allowing 10,000 trials improves the default response to 3.008547, 3.001221, 1.015873 with an optimum of -16.98405. The simulated annealing routine will also eventually solve the OPTIVAR example with reasonable accuracy giving a solution of 2.967033, 3.008547, 1.001221 with an optimum of -16.99762 after 10,000 trials. The evolutionary programming routine again eventually solves the OPTIVAR example giving a solution of 2.998779, 2.967033, 1.001221 with an optimum of -16.99769 after 10,000 trials. The dynamic hill climber is again quite good on this problem giving a solution of 3.009134, 2.996900, 1.002761 with an optimum of -16.99715 after 1120 evaluations. The evolutionary strategy with 10,000 trials gives 3.008487, 2.992901, 1.007102 with an optimum of -16.99278.

### 8.14.4. Bump Problem

The bump problem[16] is held in the routines "bumpfun.f" and "bumpcon.f" which should be used in conjunction with the macro file "bump.run" to carry out the two-dimensional bump analysis. The correct answer to this problem is 0.3650612 at (1.593163, 0.4705879) which is defined by the constraint boundaries. There is another similar peak of 0.2740178 at (0.4751587, 1.577826) while the major maximum within the boundaries has a height of 0.2628957 at (3.087255, 1.517143). No method gets consistently close to the true optimum, several heading for the interior point, with some completely thrown by the starting point which has zero slope in all directions. The "Control_Optimization Maximize" option must be used *before* solving this problem otherwise a minimum is found and this is where the problem starts from !

### 8.14.5. Colville Function Problem

The Colville function problem is defined as:

$$\text{minimize} 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2$$

$$+10.1((x_2 - 1)^2 + (x_4 - 1)^{2)} + 19.8(x_2 - 1)(x_4 - 1)$$

where $-10 \le x_i \le 10, i = 1, 2, 3, 4$; with the global solution (1,1,1,1) and $f(1, 1, 1, 1) = 0$. The typical solution found by the original Genocop system in 1,000,000 generations is (0.983055, 0.966272, 1.016511, 1.033368) with $f = 0.001013$, whereas a typical solution returned by the new version in 10,000 generations only, i.e., 1% of the original time, is (1.000581, 1.001166, 0.999441, 0.998879) with $f = 0.0000012$. This problem was submitted by Zbigniew Michalewicz, E-mail: zbyszek@mosaic.uncc.edu . This is quite hard for the GA but APPROX does it exactly in 150 steps !

### 8.14.6. John Holland's Royal Road Problem

John Holland's Royal Road problem[17] is designed to be easy for GA's to deal with and hard for other methods. It turns out the GA's find it pretty tough but that some other methods do it rather well. It forms another test problem and is held in the files "jhrr.c" and "jhrrfun.f" which should be used in conjunction with the data file "jhrr.d". A maximum is the goal and a value of 13.0 is possible at best, see the macro file "jhrr.run" in "cmds". The sources for this problem, which include a full description are in "jhrr_src". This example shows how to interface a problem written in C to the fortran of the main package.

# CHAPTER 9

## Direct Sub-routine Access to Search Engines

This chapter sets out the method by which the search engines of the **OPTIONS** system may be directly accessed as sub-routines by user generated code. It assumes that the user understands the basic structure of optimization and has used such methods before (e.g., by using the NAg methods[15] or those in the Numerical Recipes cook-book)[4]. The process involves calling the routine OPTDBS (optdbs_ in C) to create a suitable **OPTIONS** data-base followed by calls to the routine OPTNS (optns_ in C) to run searches. Additionally, once OPTDBS has been called, the interactive menu driven version of **OPTIONS** can be invoked either before or after calling OPTNS by a simple call to the routine OPTCTL (optctl_ in C). This routine does not require any arguments and simply starts an interactive session which can be exited by using the "Main_System" command at any point, after which control returns to the user's own program. OPTCTL allows the user to probe the internal **OPTIONS** data-base, modify its contents (including the values of the various **OPTIONS** control parameters mentioned in earlier chapters) or to initiate searches. If response surface modelling is required using direct subroutine access this can be switched on and off via the routine OPTRSS (optrss_ in C).

### 9.1. OPTDBS - data-base construction utility

OPTDBS is a fortran subroutine (which can be called by C codes as optdbs_ on unix systems) that takes a bare minimum description of an optimization problem and constructs the relevant **OPTIONS** database. Its argument list consists of the elements described in the following sub-sections. Note that the order in which variables and constraints are declared should be maintained for the associated limits, etc., applied to these quantities. Also that all variables within the **OPTIONS** package **must** be declared and used in upper-case character strings.

#### 9.1.1. OPTDBS Input Arguments

OLEVEL
: A flag that indicates the output level (0 for minimal output,1 for partial output and 2 for full output). An INTEGER variable (pointer to a long int in C).

DNULL
: A value used to indicate that a bound or limit is not set, i.e., this argument should be set to a non-zero numeric value which is different to any of the bounds or limits to be used. A DOUBLE PRECISION variable (pointer to a double in C).

NVRS The number of design variables (must agree with the number declared to OPTNS and must be at least one). Note that if arrays of design variables are used NVRS is the *total* number of elements in all the arrays added together. An INTEGER variable (pointer to a long int in C).

NDVRS
: The maximum number of discrete values per design variable specified in the array DVARS (must be at least one, which indicates that the array DVARS is not in use). Note that this facility is used to indicate that particular design variables can only take specific, pre-defined values, i.e., this might be used to indicate thicknesses of metal plates that only come in stock sizes or values of material properties that have known but discrete values. If all the variables in the problem may take arbitrary values between the upper and lower bounds set this to one. An INTEGER variable (pointer to a long int in C).

NCONS
: The number of constraints (must agree with the number declared to OPTNS and must be at least one so as to prevent compilation errors). Note that if arrays of constraints are used NCONS is the *total* number of elements in all the arrays added together. If the problem being searched is unconstrained then set this argument to one, set the single element in the CNAM array to be a string of 10 blanks

and set both upper and lower limits equal to DNULL. (N.b., when dealing with unconstrained problems the routine OPTCON must still be supplied, but should simply return without doing anything.) An INTEGER variable (pointer to a long int in C).

NPARAM
: The number of parameters that the user wishes to refer to in OPTFUN and OPTCON. Note that if arrays of parameters are used NPARAM is the *total* number of elements in all the arrays added together. If the problem being searched does not need parameters then set this argument to one, set the single element in the PNAM array to be a string of 10 blanks and set the single array element of PARAMS to DNULL.

VNAM
: The name strings used in the user supplied problem code calls to GETVV/D for getting the design variables. A CHARACTER*10 array of NVRS variables (pointer to a char array of 10 character strings in C, together with a long int at the end of the main arguments indicating the character string lengths, i.e., 10L). If a vector is being used then set all elements of the array to have the same name and these will be bound together and the third vector array subscript used to access the variable within OPTFUN and OPTCON. In such circumstances these items must be contiguous in the the VNAM array.

CNAM
: The name strings used in the user supplied problem code calls to SETVV/D for setting the constraints. A CHARACTER*10 array of NCONS variables (pointer to a char array of 10 character strings in C, together with a long int at the end of the main arguments indicating the character string lengths, i.e., 10L). If a vector is being used then set all elements of the array to have the same name and these will be bound together and the third vector array subscript used to access the variable within OPTCON. In such circumstances these items must be contiguous in the the CNAM array.

PNAM
: The name strings used in the user supplied problem code calls to GETVV/D for accessing the parameters. A CHARACTER*10 array of NPARAM variables (pointer to a char array of 10 character strings in C, together with a long int at the end of the main arguments indicating the character string lengths, i.e., 10L). If a vector is being used then set all elements of the array to have the same name and these will be bound together and the third vector array subscript can then be used to access the variable within OPTFUN and OPTCON. In such circumstances these items must be contiguous in the the PNAM array.

ONAM
: The name string used in the user supplied problem code calls to SETVV/D for setting the objective function. A CHARACTER*10 variable (pointer to a char of 10 characters length in C, together with a long int at the end of the main arguments indicating the character string length, i.e., 10L).

LVARS
: The lower bounds on the design variables which may be set to DNULL to indicate that a bound is not set (n.b., if any design variables share a common name they must all be set or all null). A DOUBLE PRECISION array of NVRS variables (pointer to an array of doubles in C).

UVARS
: The upper bounds on the design variables which may be set to DNULL to indicate that a bound is not set (n.b., if any design variables share a common name they must all be set or all null). Each element of UVARS must either be null or greater than the equivalent element of LVARS. A DOUBLE PRECISION array of NVRS variables (pointer to an array of doubles in C).

DVARS
: The discrete values of the design variables which may be set to DNULL to indicate a null value (n.b., if any null values are set for a design variable these must be at the end of the column for that variable). Each element of DVARS must either be null or greater than the previous element of that column, i.e., they must be in order. A DOUBLE PRECISION array of NVRS by NDVRS variables (pointer to a two dimensional array of doubles in C, n.b., notice that that in C the packing order of arrays is reversed and so this array is declared as NDVRS by NVRS in C).

LCONS
: The lower limits on the constraints which may be set to DNULL to indicate that a limit is not set (n.b., if any constraints share a common name they must all be set or all null). A DOUBLE PRECISION array of NCONS variables (pointer to an array of doubles in C).

UCONS

The upper limits on the constraints which may be set to DNULL to indicate that a limit is not set (n.b., if any constraints share a common name they must all be set or all null). Each element of UCONS must either be null or greater than/equal to the equivalent element of LCONS. A DOUBLE PRECISION array of NCONS variables (pointer to an array of doubles in C).

PARAMS

The values of the parameters which may be set to DNULL to indicate that a parameter is not set (n.b., if any parameters share a common name they must all be set or all null). A DOUBLE PRECISION array of NPARAM variables (pointer to an array of doubles in C).

### 9.1.2. OPTDBS Output Arguments

ERROR

The error flag. A LOGICAL variable (pointer to a long int in C).

### 9.1.3. OPTDBS Use

The use of this routine is illustrated by the first half of the program "beam1s.f" ("beam1sc.c" for C version), which may be used to run the example problem discussed in Chapter 5 in conjunction with the routines "beam1fun.f" and "beam1con.f" discussed there. When used in this way there is no direct user interaction with the **OPTIONS** suite. The source of "beam1s.f" is provided in Appendix K and the source of both this routine and the C equivalent are supplied with the **OPTIONS** package.

Notice that the program is quite short but that it not only sets up the required data-base but also runs all three of the searches described in Chapter 5, returning identical results to those given there. Basically, the program defines the size of the problem (in the PARAMETER statements), then sets up the design variables (using strings that match those in the routines "beam1fun.f" and "beam1con.f"), the constraints (again using strings that match those in the routines "beam1fun.f" and "beam1con.f") and the parameters (also using strings that match those in the routines "beam1fun.f" and "beam1con.f"). Note the use of the PARAMETER DNULL to indicate unset limits. This item may take any value except one of those actually used as a limit (it is also the case that if the lower and upper limits are identical an equality constraint is created). Finally, the objective function name is set (also using a string that matches that used in "beam1fun.f"). These various elements are then passed to OPTDBS as arguments and a matching data-base is set up automatically. Notice that parameters which do not directly form part of the search process (such as the value of $\nu$, the Poisson's ratio) do not need to appear in this process. This is acceptable because the routines "beam1fun.f" and "beam1con.f" described there refer to these quantities via calls with explicit default values (GETDEF, GETDFD, SETDEF, SETDFD). Thus when used with OPTDBS and OPTNS these defaults would be accessed instead of taking values from the **OPTIONS** data-base. This approach can be used at the same time as declaring parameters using OPTDBS.

If, when OPTDBS is called, it returns with the error flag set then the user's definitions will not be valid and should be modified to correct the error indicated by output from the routine. The amount of such diagnostic information is controlled by OLEVEL which should be set to two or more while developing the calling program.

Sometimes when using the API a user will want to manipulate variable dimensions after calling OPTDBS. For example the user may wish to pre-load the trace arrays with data when doing response surface modelling. When OPTDBS is run by deafult the OPT_TRACE and OPT_CTRACE arrays are set to have dimensions of (1,1,1) and these then need changing to suit the problem being dealt with. To change the trace array dimensions the user must call the routine "CHGDIM" after OPTDBS and before attempting to load such data. To set the dimensions of OPT_TRACE use:

```
CALL CHGDIM('OPT_TRACE',NVARS+1,IY,MZ,NOROOM,ERROR)
```

This routine is described in the headers sections of the "docs" subdirectory but basically the three integers arguments following the array name set the array sizes. So NVARS would be the number of variables (with +1 to allow for the objective function values). Typically IY would be 1 and MZ would be 1000 but these are up to the user. NOROOM and ERROR are logical flags that indicate if the change failed either because the data-base cannot provide the desired space or some other error (such as the variable does not exist). The equivalent call for the constraints is

```
      CALL CHGDIM('OPT_CTRACE',NEQ+NINEQ+NRANGE+1,MAX(NVARS,3)
     1 ,MZ,NOROOM,ERROR)
```

Here NEQ is the number of equality constraints, NINEQ the number of inequality constraints and NRANGE the number of range constraints (i.e., both upper and lower inequality constraints applied to the same quantity). MZ would again normally be 1000. Having made such changes it is wise to run a tidy-up on pointers in the data-base as follows:

```
C
C     reset common blocks etc
C
      CALL OPTPNT('Obj_Funct',.FALSE.,ERROR)
      IF(ERROR)CALL OPTERR(2)
      CALL OPTPNT('Opt_Var',.FALSE.,ERROR)
      IF(ERROR)CALL OPTERR(2)
      CALL OPTPNT('Constraint',.FALSE.,ERROR)
      IF(ERROR)CALL OPTERR(2)
```

Again ERROR is a logical flag indicating that the action failed if .TRUE. This reset process just ensures that internal **OPTIONS** pointers are not confused by direct user intervention. Such array resizing can be carried on any of the variables defined by OPTDBS that may be required by the user.

### 9.2. OPTNS - search engine utility

OPTNS is a fortran subroutine (which can be called by C codes as optns_ on unix systems) that calls on a previously set-up data-base description of an optimization problem and applies one of the **OPTIONS** search engines to it (it may also be used to simply evaluate the user supplied problem code if desired). Its argument list consists of the elements described in the following sub-sections.

#### 9.2.1. OPTNS Input Arguments

OLEVEL

A flag that indicates the output level (0 for minimal output,1 for partial output, 2 for full output and 3 for full output plus reports during the search process). An INTEGER variable (pointer to a long int in C).

OMETHD

The search method to use which has the following meaning :-

| | |
|---|---|
| 0.0 | to just evaluate the user's problem code at the point specified |
| 1.1 | for OPTIVAR routine ADRANS |
| 1.2 | for OPTIVAR routine DAVID |
| 1.3 | for OPTIVAR routine FLETCH |
| 1.4 | for OPTIVAR routine JO |
| 1.5 | for OPTIVAR routine PDS |
| 1.6 | for OPTIVAR routine SEEK |
| 1.7 | for OPTIVAR routine SIMPLX |
| 1.8 | for OPTIVAR routine APPROX |
| 1.9 | for OPTIVAR routine RANDOM |
| 2.1 | for user specified routine OPTUM1 |
| 2.2 | for user specified routine OPTUM2 |
| 2.3 | for NAg routine E04UCF |
| 2.4 | for bit climbing |
| 2.5 | for dynamic hill climbing |
| 2.6 | for population based incremental learning |
| 2.7 | for numerical recipes routines |
| 2.8 | for design of experiment based routines |
| 2.8 | for design of experiment based routines (but without function calls) |
| 3.11 | for Schwefel library Fibonacci search |

| | |
|---|---|
| 3.12 | for Schwefel library Golden section search |
| 3.13 | for Schwefel library Lagrange interval search |
| 3.2 | for Schwefel library Hooke and Jeeves search |
| 3.3 | for Schwefel library Rosenbrock search |
| 3.41 | for Schwefel library DSCG search |
| 3.42 | for Schwefel library DSCP search |
| 3.5 | for Schwefel library Powell search |
| 3.6 | for Schwefel library DFPS search |
| 3.7 | for Schwefel library Simplex search |
| 3.8 | for Schwefel library Complex search |
| 3.91 | for Schwefel library two-membered evolution strategy |
| 3.92 | for Schwefel library multi-membered evolution strategy |
| 4 | for genetic algorithm search |
| 5 | for simulated annealing |
| 6 | for evolutionary programming |
| 7 | for evolution strategy |

A DOUBLE PRECISION variable (pointer to a double in C).

DIRCTN
: The search direction (in the range +/-2). The optimizers try to minimize the objective function if this argument is -1, maximize it if is +1, minimize the log of the function if it is -2 or maximize the log if it is +2. An INTEGER variable (pointer to a long int in C).

NITERS
: The number of iterations (calls to the objective function) to use. This variable may not always be adhered to depending on the method used. Must be at least one. An INTEGER variable (pointer to a long int in C).

NVRS The number of design variables (must agree with the number declared to OPTDBS and must be at least one.) An INTEGER variable (pointer to a long int in C).

NCONS
: The number of constraints (must agree with the number declared to OPTDBS and must be at least one so as to prevent compilation errors.) If the problem being searched is unconstrained then set this argument to one, see notes on OPTDBS arguments for further details for unconstrained problems. An INTEGER variable (pointer to a long int in C).

### 9.2.2. OPTNS Input / Output Arguments

VARS The design variable vector. Contains the starting point for the search at input and the end point at output. A DOUBLE PRECISION array of NVRS variables (pointer to an array of doubles in C).

### 9.2.3. OPTNS Output Arguments

CONS The constraint vector at the end of the search, which should be compatible with the limit values set up by the call to OPTDBS if the search has been successful. A DOUBLE PRECISION array of NCONS variables (pointer to an array of doubles in C).

OBJFN
: The objective function at the end of the search. A DOUBLE PRECISION variable (pointer to a double in C).

NCALLS
: The number of function calls actually used in the search. An INTEGER variable (pointer to a long int in C).

ERROR
: The error flag which will be .FALSE. (0 in C) if no problems arose during the search. A LOGICAL variable (pointer to a long int in C).

### 9.2.4. OPTNS Use

The use of this routine is illustrated by the second half of the program "beam1s.f" ("beam1sc.c" for C version). It is almost always used after a call to OPTDBS has been used to set up the relevant, problem specific, **OPTIONS** data-base. When used in this way there is no direct user interaction with the **OPTIONS** suite. The source of "beam1s.f" is provided in Appendix K and the source of both this routine and the C equivalent are supplied with the **OPTIONS** package.

Notice that the program is quite short but that it not only runs all three of the searches described in Chapter 5, returning identical results to those given there, but also sets up the required data-base before calling OPTNS. Basically, having set-up the data-base via a call to OPTDBS, OPTNS is called with a series of arguments that set the level of output required, the search method to be used, the direction of search (minimization, maximization, etc.,) and the number of search steps, followed by parameters indicating the problem size (which must match those already used with OPTDBS) and the start point for the search. On return the start point is up-dated to the end point, constraint values are returned as is the final objective function, the number of function calls actually made and an error flag. If the error flag is not set this means that a successful search has been carried out (n.b., this does *not* mean that an optimum has been found, merely that the search ended normally). If a user requires to be assured that the returned point is a true optimum, additional independent checks must be made on the final result returned by **OPTIONS**. The error flag may also be set on return if the user's input is not valid which should then be modified to correct the error indicated by output from the routine. The amount of such diagnostic information is controlled by OLEVEL which should be set to two or more while developing the calling program.

Note that if non-default values are required for any of the optimizers control parameters these variables have to be set using the SETVV routines before calling OPTDBS. Moreover, the additional parameters mentioned in the subsection entitled "Optimizing Optimizer Control Paramters" from the previous chapter, such as OVR_MAND and OVR_PENAL, can be used to control those optimizer settings that would normally be made by command line input, see the previous chapter for further details.

## 9.3. OPTRSS - response surface setup utility

OPTRSS is a fortran subroutine (which can be called by C codes as optrss_ on unix systems) that allows the user to switch between response surface modelling of a problem and normal access to that problem (response surface modelling is described in detail in the relevant chapter of this manual). This routine should not be called before the user has set up the problem being worked on via OPTDBS and produced data from which the response surfaces can be constructed via at least one search using OPTNS (a good way of setting up such data is via the design of experiments optimizer). This data is held in the data-base variable OPT_TRACE for the objective function model and in OPT_CTRACE for the constraint model. These arrays are automatically sized and filled with data by running any **OPTIONS** search, provided the items exist at run-time, which they should do if OPTDBS has been called. If the data to be used is to be inserted by calls to SETVV etc., rather than via a search, the arrays can be re-sized after OPTDBS has been called - this process is described in the earlier section on "OPTDBS Use".

The OPTRSS argument list consists of the elements described in the following sub-sections.

### 9.3.1. OPTRSS Input Arguments

OPTRSS provides direct subroutine access to the response surface modelling system for those using the **OPTIONS** API.

OBJMOD

A flag variable which should be set to 1.0 if a Shepard response surface model should be constructed and used for the objective function in the problem, 2.1 to 2.3 for a Radial Basis Function model (2.1 for linear, 2.2 for thin plate and 2.3 for cubic splines), 3.1 to 3.5 for a polynomial regression model (3.1 for mean, 3.2 for first order, 3.3 for first order plus squares, 3.4 for first order plus products (cross-terms), 3.5 for all second order and 3.6 for second order plus cubes), 4.1 for a Stochastic Process Model, 4.2 for the root mean square error of the Stochastic Process Model, 4.3 for the expected improvement of the Stochastic Process Model (4.31 gives constrained expected improvement using a constraint SPM, 4.32 gives probability of feasibility using a constraint SPM and 4.33 gives probability of improvement), 4.4 for a twin objective Pareto front Stochastic Process Model approach (4.41 gives a model based on the combined probability of improvement, including augmentations to the front, 4.42 is combined probability of improvement but requiring all new points to dominate a memeber of the old front, 4.43 uses an average between these two measures while 4.44 to 4.46 are similar but based on the expected improvement rather than the probability of improvement) and 0.0 if the underlying user supplied function is to be called. A REAL variable (pointer to a float in C).

CONMOD

A flag variable which should be set to 1.0 if a Shepard response surface model should be constructed and used for all the constraints and their limits in the problem, 2.1 to 2.3 if Radial Basis Function models are to be used (2.1 for linear, 2.2 for thin plate and 2.3 for cubic splines), 3.1 to 3.5 for a polynomial regression model (3.1 for mean, 3.2 for first order, 3.3 for first order plus squares, 3.4 for first order plus products (cross-terms), 3.5 for all second order and 3.6 for second order plus cubes) and 0.0 if the underlying user supplied function is to be called. A REAL variable (pointer to a float in C).

UPDATE

A flag that indicates the response surface data-sets in OPT_TRACE and OPT_CTRACE are to be automatically updated at the end of any searches by calls to the user's full problem code if .TRUE. (set .FALSE. if not or to turn of updating of the arrays). A LOGICAL variable (pointer to a long int in C, 0 for .FALSE., 1 for .TRUE.). N.b., before attempting to call the user's own code the arrays are scanned to see if the current point is already present. If it is the point is randomly perturbed in an attempt to improve the quality of the data-set in the area of interest. Note also that when a model is updated by making a call to the user's code then the data produced is added to the trace arrays automatically but the response surface model coefficients themselves are *not* re-calculated. It remains the user's responsibility to do this by invoking the RSM system by calling OPTRSS as though for the first time. This allows users to choose when such re-calculations occur as they can be time-consuming.

### 9.3.2. OPTRSS Output Arguments

ERROR

The error flag. A LOGICAL variable (pointer to a long int in C). This flag is set to indicate that the response surface models could not be set up, most commonly because OPT_TRACE or OPT_CTRACE did not exist or contain sufficient data for either the objective function or constraint response surface models to be constructed.

### 9.3.3. OPTRSS Use

To invoke response surface modelling OPTRSS is called after a space filling search has been executed (via the routines OPTDBS and OPTNS). Following this, any calls to OPTNS will carry out searches on the response surface meta-models and only invoke the user's own code once at the end of each search (and then only if UPDATE has been set .TRUE.). To return to searches on the full user code OPTRSS should be called a second time with the input flags set as zeros and .FALSE..

# CHAPTER 10

## Response Surface Modelling

For many problems the evaluation of a new design point may be computationally expensive. This tends to prohibit extended searches using optimization methods[18, 19]. In such circumstances "response surface modelling" can be of value. The technique consists of constructing a meta-model of the user's problem based on a few, often pre-computed, results. This meta-model is then searched in lieu of the full problem. Then, when a good design is found, the full problem code is used to evaluate just this design. If there are significant differences between the full code and the meta-model at this point the new data can be added to that used to construct the meta model and the search process carried out again. This sequence can then be repeated as many times as desired, gradually improving the quality of the design and the meta-model in the region of the best designs found. Of course, the meta-model can only reflect the data used to form it and so if this does not encompass the best areas of the search space, promising designs will be missed. This difficulty can be mitigated against to some extent by using an expected improvement criterion[19] when choosing where to evaluate new points, since this balances improvements in the design against improvements in the accuracy of the meta-model.

If the user has a cheap approximation to the desired objective function then it is possible to build a meta-model of the differences between the full code and the approximate model. This may then be used as a correction to the cheap model and this combined or "fused" model used instead of the response surface of the expensive code alone.

In all circumstances a search on the full problem code is to be preferred if possible and response surface results should be used with caution. **Extreme care** should be taken not to confuse response surface model results with those from the real problem. This chapter of the manual describes the **OPTIONS** response surface modelling system.

### 10.1. Response Surface Model construction

**OPTIONS** allows for two separate response surface models to be constructed at the same time, one for the objective function and one (or more) for the constraints. The objective function model is constructed using data in the data-base variable OPT_TRACE while the constraint model uses that in OPT_CTRACE. These arrays are automatically sized and filled with data if present during any **OPTIONS** search (they are always present when using the API interface routines OPTDBS or optdbs_). Thus, a good way to build the data for the response surface meta-models is to use a space filling search before invoking these facilities. Of the space filling routines within **OPTIONS** the Design of Experiments method[11] is best suited to give even coverage (and this can be achieved in parallel if a suitable parallel processing environment exists). Within this optimizer there are a number of ways of selecting the points to analyse, including such methods as Latin hypercubes and $LP_\tau$ arrays, as well as pure random searches and simple full-factorial approaches. Alternatives are any of the population based stochastic methods or Dynamic Hill Climbing[9]. It is also possible to select points by hand or even to modify designs produced by the Design of Experiments optimizer (its methods can be invoked to fill the trace arrays without calling the user's code by using the "Test_DoE_Optimizer" command and these can then be used as a candidate list for the Design of Experiments optimizer).

Having filled one or other (or both) of these two arrays with data from the full problem code, response surface modelling can be invoked from the **OPTIONS** main menu using the "Control_Optimization" command with the "RSM_model_on" argument. This action can be cancelled with the "USER_model_on" or "Reset" arguments (the later resets other facilities as well, see command reference section for details). Alternatively, if the API interface routines are being used the fortran subroutine OPTRSS (which can be called by C codes as optrss_ on unix systems) may be used instead.

To allow for the possibility that some of the data in the OPT_TRACE array may be the results of failed calculations (this may arise, for example, when a space filling search enters regions where the user's code fails to converge) the user may indicate an outer bound for acceptable values of objective function data. This bound is specified in the variable OBJ_BAD_PT and then any point where the objective function is greater than this (for minimization) or less (for maximization) is ignored when building the response surface for the objective. Similarly CST_BAD_PT may be used to provide a threshold for constraint values (taken to be the the lower limit of acceptable values when range constraints are used).

Note that when a response surface model is invoked to replace the objective function or the constraints then the user supplied routines OPTFUN and OPTCON are not used, respectively. Instead the response surface models are called to evaluate these quantities; this is reflected in the information reported via the Show_Optimization command. If the user code is supplied as the combined routine OPTFNC then response surface models must be invoked for both the objective function *and* the constraints if calls to OPTFNC are to be avoided, i.e., both OPT_TRACE and OPT_CTRACE must be included in the data-base and filled with data before invoking response surface modelling.

If data fusion is invoked both OPTFUN and the response surface model are used to provide results for the objective function.

### 10.1.1. Response Surface Models

Various sorts of meta-model may currently be constructed within **OPTIONS**. The first of these is very simple, fast to use but limited in sophistication. It is based on a simple first-order cubic Shepard weighting scheme[20, 21] where the estimated value of a function $\hat{y}_{n+1}$ at some new point $n+1$ in a $k$ dimensional design space, $\mathbf{x}_{n+1} = (x_{1,n+1}, x_{2,n+1}, \cdots, x_{k,n+1})^T$, is estimated from the previous $n$ values using

$$\hat{y}_{n+1} = \sum_{i=1}^{m} w_i(\mathbf{x}_{n+1}, \mathbf{x}_i) Q_i(\mathbf{x}_{n+1}) \quad .$$

Here, the data points are sorted in Euclidean distance order from $\mathbf{x}_{n+1}$ and $m$ is the number of data points used in the estimate such that $3 \le m \le n$. The weights $w_i$ are given by

$$w_i(\mathbf{x}_{n+1}, \mathbf{x}_i) = \frac{\left( \dfrac{|\mathbf{x}_{n+1} - \mathbf{x}_m|}{|\mathbf{x}_{n+1} - \mathbf{x}_i|} - 1 \right)^3}{\displaystyle\sum_{i=1}^{m} \left( \dfrac{|\mathbf{x}_{n+1} - \mathbf{x}_m|}{|\mathbf{x}_{n+1} - \mathbf{x}_i|} - 1 \right)^3} \quad .$$

The functions $Q_i$ can be of arbitrary order (Shepard used 2nd order) and here, to save on computing effort, they are taken as first order such that

$$Q_i(\mathbf{x}_{n+1}) = y_i + c_i \sum_{j=1}^{k} (x_{j,n+1} - x_{j,i})$$

where the coefficient in the first order term, $c_i$, provides local gradient information (if it is set to zero a zeroth order approximation results with the response surface having locally zero gradients at each data point). This coefficient is calculated from the gradient between the point $i$ and its nearest neighbour $l$ such that

$$c_i = \frac{(y_l - y_i)}{\displaystyle\sum_{j=1}^{k} (x_{j,l} - x_{j,i})} \quad .$$

This ensures that the local gradient in each of the $k$ dimensions is the same and that a plane through the point $i$ with this local gradient would pass through the nearest neighbour point $l$. This is the simplest linear form that can be specified for the $Q_i$'s.

Within **OPTIONS** the number of data points used, $m$, is related to the dimensions of the problem so that

$$m = \min(n, 2^k + 1) \quad .$$

When set up in this way the $2^k + 1$th point has a weight of zero, and all points further from $\mathbf{x}_{n+1}$ make no contribution to the estimate. This model interpolates within the remaining $2^k$ points but models curvature with limited accuracy. It is, however, extremely easy to set up and fast to apply, readily dealing with up to a thousand data points with no numerical difficulties. It does not provide estimates of the variance of the estimate and so an expected improvement criterion cannot be used with this model to decide on where to add points when seeking an optimum. If searches are made using this model then the best optimal position is used when it is updated. If this point is already in the data-set a random point within the hyper-cube defined by the distance to the point at $\mathbf{x}_m$ is used instead.

The next most sophisticated model available in **OPTIONS** uses a polynomial interpolation approach. In this model the objective function or each of the constraints and limits are approximated by the same form of equation

$$\hat{y}_{n+1} = \sum_{i=1}^{l} a_i \phi_i(\mathbf{x}_{n+1})$$

where the $a_i$ are coefficients derived to ensure that the model fits the data and $l \leq n$ is the number of regressing functions (if $l = n$ an interpolation results). The functions $\phi$ can take many forms; **OPTIONS** currently offers four variants. These are simple mean value ($\phi_1(\mathbf{x}) = 1$), mean plus linear terms ($\phi_{i+1}(\mathbf{x}) = x_i$), mean plus linear terms plus squares ($\phi_{i+k+1}(\mathbf{x}) = x_i^2$), mean plus linear plus cross terms ($\phi_{i+k+1}(\mathbf{x}) = x_i x_j, j = i+1 \rightarrow k$), mean plus linear plus all second order terms (squares and cross terms) ($\phi_{i+2k+1}(\mathbf{x}) = x_i^2, x_i x_j, j = i+1 \rightarrow k$), and this plus cubes ($\phi_{i+\frac{(k+1)^2+k+1}{2}}(\mathbf{x}) = x_i^3$). The $a_i$ are given by solving

$$\left[\phi(\mathbf{x})\right]\{a_i\} = \left\{y_j\right\} \quad .$$

In practise this is carried out by singular value decomposition in conjunction with the objective function, constraint and limit values in the data arrays to derive the sets of $a_i$ coefficients. The objective function and each constraint and limit have their own set of coefficients. **OPTIONS** currently only allows 500 data points to be used in such models and if more points are available a subset of them is taken. This subset is based mainly on the most recent results (bottom end of the data array) augmented by a random selection from the rest of the array (10%) plus any particularly good results in the remaining results. Duplicate points are rejected. This model only interpolates within the points used if as many functions are used as data points and this will only happen if $\frac{(k+1)^2+3k+1}{2}$ or fewer data points are used. Otherwise the model regresses the data and the user should be aware that since the polynomial forms are applied globally this model is unable to represent complex trends in the data. Its primary use is where the data is known to be corrupted with significant noise and at the same time a simplistic model will suffice. It is fairly easy to set up and fast to apply, readily dealing with up to 500 data points with no numerical difficulties. It does not currently provide estimates of the variance of the estimate and so an expected improvement criterion cannot be used with this model to decide on where to add points when seeking an optimum. If searches are made using this model then the best optimal position is used when it is updated. If this point is already in the data-set a random point within the hyper-cube defined by the distance to the point at $\mathbf{x}_m$ is used instead, where again $m = \min(n, 2^k + 1)$.

After this the next most sophisticated model available in **OPTIONS** uses a radial basis function approach. In this model the objective function or each of the constraints and limits are approximated by the same form of equation

$$\hat{y}_{n+1} = \sum_{i=1}^{n} a_i \phi(|\mathbf{x}_{n+1} - \mathbf{x}_i|)$$

where the $a_i$ are coefficients derived to ensure that the model interpolates the data and $|\mathbf{x}_{n+1} - \mathbf{x}_i|$ measures the Euclidean distance from the location of interest to each of the data points in the OPT_TRACE or OPT_CTRACE arrays. The function $\phi$ can take many forms; **OPTIONS** currently offers three variants. These are linear spline ($\phi(x) = x$), thin plate spline ($\phi(x) = x^2 \log_{10}(x)$) and cubic spline ($\phi(x) = x^3$). The $a_i$ are given by

$$\{a_i\}^{\mathrm{T}} = \left\{y_j\right\}^{\mathrm{T}} \left[\phi\left(|\mathbf{x}_j - \mathbf{x}_i|\right)\right]^{-1} \quad .$$

In practise the LU decomposition of the matrix of cross Euclidean distances between the points in the data arrays is formed and this is then used in conjunction with the objective function, constraint and limit values in the data arrays to derive the sets of $a_i$ coefficients. The objective function and each constraint and limit have their own set of coefficients. **OPTIONS** currently only allows 500 data points to be used in such models and if more points are available a subset of them is taken. This subset is based mainly on the most recent results (bottom end of the data array) augmented by a random selection from the rest of the array (10%) plus any particularly good results in the remaining results. Duplicate points are rejected. This model interpolates within the points used. It is fairly easy to set up and fast to apply, readily dealing with up to 500 data points with no numerical difficulties. It does not currently provide estimates of the variance of the estimate and so an expected improvement criterion cannot be used with this model to decide on where to add points when seeking an optimum. If searches are made using this model then the best optimal position is used when it is updated. If this point is already in the data-set a random point within the hyper-cube defined by the distance to the point at $\mathbf{x}_m$ is used instead, where again $m = \min(n, 2^k + 1)$.

Normally radial basis function models interpolate the data set. However, if they are set up around a linear regression line and a reduced basis is used they can be made to carry out regression. To do this a fit of the form

$$\hat{y}_{n+1}(\mathbf{x}_{n+1}) = a_0 + \sum_{i=1}^{k} a_i x_{i,n+1} + \sum_{j=1}^{N\le n} b_j \phi\left( |\mathbf{x}_{n+1} - \mathbf{x}_j| \right)$$

is used where $N$ is the number of basis functions used and the $a_i$ and $b_j$ are solved using

$$y_l = a_0 + \sum_{i=1}^{k} a_i x_{il} + \sum_{j=1}^{N\le n} b_j \phi\left( |\mathbf{x}_l - \mathbf{x}_j| \right) \qquad \text{for } \; l = 1, \cdots, n$$

together with the following additional constraints

$$\sum_{j=1}^{N\le n} b_i = 0$$

and

$$\sum_{j=1}^{N\le n} b_j x_{i,j} = 0 \qquad \text{for } \; i = 1, \cdots, k.$$

If the full set of functions is used ($N = n$) this reverts to the normal radial basis function model (albeit with better conditioning because the radial basis part of the model is only used to fit the corrections to the regression line - this can be used to advantage if there are numerical stability problems with the normal model). When fewer and fewer functions are used this progressively regresses the model more and more, so that, when no basis functions are involved at all, it becomes a simple linear regression fit. In **OPTIONS** this approach is only currently available for cubic splines.

**OPTIONS** uses a Null Hypothesis test to decide the degree of regression applied to the radial basis unless the user over-rides this manually. This test seeks to prevent over-fitting by starting with no basis terms and continually adding more until the convergence fails to improve sufficiently. It is vital to note that the basis functions are added in the order that the points lie in the trace arrays - if random or pseudo random sequences are used to generate these this is fine - if, however, they have been calculated in some pre-defined regular pattern this will result in a very spatially biased regression model. When manually controlling the degree of regression the user does this by specifying how many functions to omit from the end of the sequence using the variables RSM_NSKIP and RSM_NCSKIP, for the objective function and constraints, respectively (all constraints are treated alike). Alternatively, the user can specify the percentage by which the Null Hypothesis can worsen as each new basis function is added before the process is halted using the variable RSM_NULL_T. By default this value is 10 so that when a basis function is added, if the Null improves or at worst only declines by 10%, another function is added, and so on. If the user wishes to monitor this process more closely the variables RSM_NULLH and RSM_CNULLH may be defined with dimensions $1, 1, n$. Then each null value is stored as the basis functions are added and these may be examined or plotted to select the precise level of regression required (note that to explore the full range of possible regressions, RSM_NULL_T should be set large, which forces all the possible bases to be used and thus complete interpolation). In all these calculations the regression equations in the radial basis function model are non-square and so are solved using singular value decomposition to satisfy the equations in a least squares sense. The tolerance used by default in this process is $5 \times 10^{-10}$ but this can be altered by the user via the variable RSM_SVDTOL.

Where a very sophisticated model is desired **OPTIONS** provides a stochastic process model using the approach described by Jones *et al*[19]. Currently this model may only be used to represent the user's objective function and for the first constraint (any remaining constraints and all limits are modelled using polynomial regression if the stochastic process model is invoked for the first constraint - the approach is very hungry on memory). In a stochastic process model the estimated value is given by

$$\hat{y}_{n+1} = \hat{\mu} + \mathbf{r}'\mathbf{R}^{-1}(\mathbf{y} - \mathbf{1}\hat{\mu}) \quad .$$

Here $\hat{\mu}$ is the mean of the previous $y_n$ function values, $\mathbf{r}'$ is a correlation vector based on $\mathbf{x}_{n+1}$ and $\mathbf{R}$ is a matrix of correlation coefficients based on the previous $\mathbf{x}_i$ data points. The correlations are defined by

$$\mathrm{Corr}[\mathbf{x}_i, \mathbf{x}_j] = \exp\left[-\sum_{h=1}^{k} 10^{\theta_h} |x_{h,i} - x_{h,j}|^{p_h}\right] + \delta_{ij} 10^{\lambda}$$

with the $i, j$th entry of $\mathbf{R}$ given by $\mathrm{Corr}[\mathbf{x}_i, \mathbf{x}_j]$ and the $i$th element of $\mathbf{r}'$ by $\mathrm{Corr}[\mathbf{x}_{n+1}, \mathbf{x}_i]$. Here $\theta_h$ and $p_h$ are unknown coefficients in the model which are found by an iterative search on a maximum likelihood function. $\lambda$ is a further, optional, unknown coefficient known as the regularization constant that can also be found by searching the likelihood. If present this causes the model to regress the data rather than to interpolate it. It thus acts as a smoothing agent in the process. The default action is to search for $\lambda$ along with the other constants; its effect can be eliminated by making it large and negative such that $10^{\lambda}$ tends to zero. Users should take great care that regularization is not used to eliminate desirable variability in their models - if it is known that data does not contain significant noise OBJ_LAMBDA should be declared and set to a large negative value (such as -8) to ensure that the model created interpolates the data used, rather than regresses it, see below. The likelihood is given by

$$\frac{1}{(2\pi)^{n/2}(\sigma^2)^{n/2}|\mathbf{R}|^{1/2}} \exp\left[-\frac{(\mathbf{y} - \mathbf{1}\mu)^T \mathbf{R}^{-1} (\mathbf{y} - \mathbf{1}\mu)}{2\sigma^2}\right]$$

where $\mu$ is the mean and $\sigma^2$ the variance of the model. (N.b., in fact we seek the minimum of -log(CLF) within **OPTIONS**.) Notice that the matrix $\mathbf{R}$ has $n^2$ elements and must be inverted and stored in this process (in practise an LU decomposition is stored and used instead) - this makes models with more than around 100-200 data points difficult to deal with, since the matrix inversion process starts to become ill-conditioned (although the ill-conditioning is to some extent mitigated by the presence of $\lambda$).

When using this model for the objective it is possible to calculate the expected value of the function at a new data point (constrained or unconstrained), its variance there or the expected improvement in the model. For constraints only the expected value of the function at a new data point or its variance may be found. The user is able to make this choice when selecting the random process model (or within the argument list when using the API version). When working with constrained expected improvement the normal expected improvement is multiplied by a probability of constraint feasibility directly using the stochastic process model of the first constraint (which is then not applied seperately - this is only possible for the first constraint). It is also possible to use the objective function and constraint SPM models together to provide a multi-objective expected improvement capability, see Chapter 11. If searches are made on the expected improvement function new points can be added to the meta-model in an optimal fashion (although it should be noted that this function will be, by definition, more multi-modal than the underlying problem). The weighting used between expected values and variance is set by default in this process as 0.5 but this can be altered by the user via the variable RSM_EIF_WT between zero and one. A value of one sets the search to maximum exploitation, i.e., it focuses solely on the expected function values, while a weight of zero sets the search to maximum exploration, i.e., a focus on reducing the global errors in the model.

When using the variance as a target for optimization the system must be set up to *minimize* the variance while when using expected improvement the system must seek to *maximize* the expected improvement. Stochastic process models are not suitable for use with large data-sets as they are time consuming to set up since constructing the model involves a sequence of matrix inversions to be carried out when choosing the model parameters $\theta_h$, $p_h$ and $\lambda$. Moreover, when large numbers of data points are used the matrix inversion process tends to become unstable, while the search for the hyper-parameters becomes highly multi-modal and easy to get wrong. The method does, however, give good statistical information on the quality of the model being built, allowing statistical validation using standardized cross-validated residuals to be tested (a warning is generated if any of these lie outside the range ±3 and if this occurs the model should be used with extreme caution).

Current best advice indicates that no more than $50k$ data points should be used when creating such models otherwise they tend to become unstable. **OPTIONS** currently only allows 500 data points to be used in such models and if more points are available a subset of them is taken. This subset is based mainly on the most recent results (bottom end of the OPT_TRACE OPT_CTRACE arrays) augmented by a random selection from the rest of the array (10%) plus any particularly good results in the remaining results (OPT_TRACE only). Duplicate points are rejected. When searches are being made to update the model, then if the point to be added is already in the data-set a random point within the hyper-cube defined by the distance to the point at $\mathbf{x}_m$ is used instead, where again $m = \min(n, 2^k + 1)$. Note also that when a model is updated by making a call to the user's function then the data produced is added to the trace arrays automatically but the response surface model coefficients themselves are *not* re-calculated. It remains the user's responsibility to do this by invoking the RSM commands as though for the first time. This allows users to choose when such re-calculations occur as they can be time-consuming.

Because the Stochastic Process Model can be so time consuming to set up and also sometimes unstable the user may manually control or save the $\theta_h$, $p_h$ and $\lambda$ coefficients in the **OPTIONS** data-base variables OBJ_THETA and OBJ_EXP which are arrays with the same dimensions as the underlying problem (i.e., 1,1,$k$) and OBJ_LAMBDA which is a scalar (and similarly CST_THETA, CST_EXP and CST_LAMBDA for the constraint model). These should be defined before first creating the model and they are then sized appropriately and filled with the relevant results from maximum likelihood searches when the model is created. On subsequent use of the model these values are adopted until they are manually cleared by the user. If only OBJ/CST_EXP and OBJ/CST_LAMBDA are created and filled with data then searches will just seek values for the $\theta_h$ coefficients with the $p_h$ and $\lambda$ values being fixed. This is sometimes of use when it is desired to leave the $p_h$ parameters at the fixed value of two giving the smoothest possible model or $\lambda$ at some large negative value to prevent any regression (i.e., forcing the model to exactly interpolate the data). Setting $p_h$ reduces the length of any search since only half as many parameters must be found to tune the likelihood function. If only OBJ/CST_LAMBDA is created and filled with data then searches will just seek values for the $\theta_h$ and $p_h$ coefficients with $\lambda$ being fixed. Since such tuning can be time consuming the user can cause all the $\theta_h$ paramters to be equal to each other and also all the $p_h$ parameters to be eqaul to each other. This means that whatever the complexity of the problem only three hyper-paramters need to be searched for. To do this the data-base variable RSM_QCK_HP must be defined and set equal to the number of design variables in the problem. This is only really sensible if these values would have been broadly similar if tuned individually and this will be problem dependant. Note finally that searching for all the parameters can be difficult and so in critical cases the user is well advised to manually study the effects on the likelihood (and the modelling) of small changes in the model parameters to see what effects they have.

To help in studies of the coefficients in the S.P. model the negation of the log of the concentrated likelihood function is stored in the variables OBJ_CLF and CST_CLF if they exist. Moreover, once the initial Stochastic Process Model is set up, if one or other of these variables is declared as the objective function and the variables OBJ/CST_THETA, OBJ/CST_EXP and OBJ/CST_LAMBDA are made the design variables it is possible to use the full power of the **OPTIONS** search system to tune the model by *minimizing* this quantity (because we have negated its logarithm). To help in this process it is also desirable to create the variables L_OBJ/CST_THET, U_OBJ/CST_THET, L_OBJ/CST_EXP, U_OBJ/CST_EXP, L_OBJ/CST_LAMB and U_OBJ/CST_LAMB which can then be used to limit the range of such searches. These variables are created automatically by the macro file "params.dbs" in the "cmds" sub-directory. Sensible limits for the $\theta$'s and $\lambda$'s are -10 and 3 while the exponent should lie between 1 and 2. If these exist before the Stochastic Process Model is built they are automatically sized and filled with appropriate data. It is then a simple matter of creating the default Stochastic Process Model to give the expected function value, changing the variable types in the user problem while tuning the model and then back again when this is completed. Note that the initial Stochastic Process Model *must* be created before this process is invoked to ensure that all the required data-structures within **OPTIONS** have been created. An example of this approach is provided as "bump.rsm_clf" in the "cmds" sub-directory.

If the stochastic process modelling fails to give sufficiently small residuals consideration should be given to transforming the data-sets being used before fitting, using either $1/y$ or $\log(y)$ operators before constructing the model (such transformations can be carried out directly on the data in the trace arrays using the **OPTIONS** Transform_Data Menu menu options of Copy_Sub_Array, Scale_and_Shift and Log10(Abs(Value)), (n.b., the update function will not automatically do this for new points which must then be manually transformed as they are added; also note that when a model is updated by making a call to the user's function then the data produced is added to the trace arrays automatically but the response surface model coefficients themselves are *not* re-calculated. It remains the user's responsibility to do this by invoking the RSM commands as though for the first time. This allows users to choose when such re-calculations occur as they can be time-consuming.)

On some occasions a user may have an approximation to the main problem being searched. In such circumstances the user may build a RSM for the differences between the expensive and cheap codes or the ratio between them (assuming there are no zeros in the cheap data). This may then be used as correction to the cheap approximation - a process commonly termed fusion. By default fusion is used to produce models of the differences. If the user wishes to model ratios the variable FUSION_TYP must be created and given a value of unity.

To carry out fusion the user must manually add the pre-computed expensive data into the OPT_TRACE array alongside those found when repeating the calculations for the cheap model. Thus OPT_TRACE contains two values for each design point (cheap followed by expensive - the user must resize the OPT_TRACE array manually by increasing its first subscript by one and then insert the expensive results alongside the existing cheap ones). Note that the OPT_TRACE array is automatically sized when carrying out searches so resizing should be carried out after computing all the data needed to form the SPM model (use the Modify_Data_Record * Dimensions command). The insertion of expensive data can be carried out by hand or by copying data from some other suitably sized array in the data-base (using the Copy_Sub_Array command in the Transform_Data sub-menu). Then when the RSM model is used the user's cheap code is called alongside the RSM to provide a corrected approximation for search, etc. This process is invoked automatically if the OPT_TRACE array contains sufficient data when the RSM is built. Note, however, that automatic data-set updating cannot be used when this process is working and so is turned off even if the user asks for it.

When using the **OPTIONS** response surface modelling system interactively the user is prompted for a modelling code for the objective function and for the constraints. These codes are 1.0 if a Shepard response surface model is to be used, 2.1 to 2.4 for a Radial Basis Function model (2.1 for linear, 2.2 for thin plate, 2.3 for cubic splines and 2.4 for cubic splines with regression via reduced bases), 3.1 to 3.6 for a polynomial regression model (3.1 for mean, 3.2 for first order, 3.3 for first order plus squares, 3.4 for first order plus products (cross-terms), 3.5 for all second order and 3.6 for second order plus cubes), 4.1 for a Stochastic Process Model, 4.2 for the root mean square error of the Stochastic Process Model and 4.3 for the expected improvement of the Stochastic Process Model (model fusion is invoked with a code of 4.1 if sufficient data is present in the OPT_TRACE array).

#### 10.1.2. OPTRSS Input Arguments

OPTRSS provides direct subroutine access to the response surface modelling system for those using the **OPTIONS** API.

OBJMOD

A flag variable which should be set to 1.0 if a Shepard response surface model should be constructed and used for the objective function in the problem, 2.1 to 2.3 for a Radial Basis Function model (2.1 for linear, 2.2 for thin plate, 2.4 for cubic splines and 2.4 for cubic splines with regression via reduced bases), 3.1 to 3.6 for a polynomial regression model (3.1 for mean, 3.2 for first order, 3.3 for first order plus squares, 3.4 for first order plus products (cross-terms), 3.5 for all second order and 3.6 for second order plus cubes), 4.1 for a Stochastic Process Model, 4.2 for the root mean square error of the Stochastic Process Model, 4.3 for the expected improvement of the Stochastic Process Model and 0.0 if the underlying user supplied function is to be called. A REAL variable (pointer to a float in C).

CONMOD

A flag variable which should be set to 1.0 if a Shepard response surface model should be constructed and used for all the constraints and their limits in the problem, 2.1 to 2.4 if Radial Basis Function models are to be used (2.1 for linear, 2.2 for thin plate, 2.3 for cubic splines and 2.4 for cubic splines with regression via reduced bases), 3.1 to 3.6 for a polynomial regression model (3.1 for mean, 3.2 for first order, 3.3 for first order plus squares, 3.4 for first order plus products (cross-terms), 3.5 for all second order and 3.6 for second order plus cubes), 4.1 for a Stochastic Process Model, 4.2 for the root mean square error of the Stochastic Process Model (only the first constraint is modelled using SPM, all remaining constraints and all limits are modelled using code 3.6) and 0.0 if the underlying user supplied function is to be called. A REAL variable (pointer to a float in C).

UPDATE

A flag that indicates the response surface data-sets in OPT_TRACE and OPT_CTRACE are to be automatically updated at the end of any searches by calls to the user's full problem code if .TRUE. (set .FALSE. if not or to turn of updating of the arrays). A LOGICAL variable (pointer to a long int in C, 0 for .FALSE., 1 for .TRUE.). N.b., before attempting to call the user's own code the arrays are scanned to see if the current point is already present. If it is the point is randomly perturbed in an attempt to improve the quality of the data-set in the area of interest. Note also that when a model is updated by making a call to the user's code then the data produced is added to the trace arrays automatically but the response surface model coefficients themselves are *not* re-calculated. It remains the user's responsibility to do this by invoking the RSM system by calling OPTRSS as though for the first time. This allows users to choose when such re-calculations occur as they can be time-consuming.

#### 10.1.3. OPTRSS Output Arguments

ERROR

The error flag. A LOGICAL variable (pointer to a long int in C). This flag is set to indicate that the response surface models could not be set up, most commonly because OPT_TRACE or OPT_CTRACE did not exist or contain sufficient data for either the objective function or constraint response surface models to be constructed.

#### 10.1.4. OPTRSS Use

To invoke response surface modelling OPTRSS is called after a space filling search has been executed (via the routines OPTDBS and OPTNS). Following this, any calls to OPTNS will carry out searches on the response surface meta-models and only invoke the user's own code once at the end of each search (and then only if UPDATE has been set .TRUE.). To return to searches on the full user code OPTRSS should be called a second time with the input flags set as zeros and .FALSE..

### 10.2. Response Surface Model use

Normally when using response surfaces, the meta-model will not be fully accurate in the region surrounding the end point of a search on the model. It is therefore normal to refine the meta-model in this region with further calls to the full problem code. Having done this a further search on the meta-model can be carried out to see if the design can be further refined. Thus a loop in which a series of searches are made after creating the response surfaces will result in only one call per loop to the full problem. However, care must be taken in such searches as it is quite common when searching near constraint boundaries for the final point of the response surface search to leave the design infeasible when the full code is invoked. This can cause some search methods to fail when trying to start a search on the refined meta-model (in such circumstances the routine SEEK, which is a Hooke and Jeeves search, may be invoked via the OPTIVAR menu with a Fiacco-McCormick penalty function - this is usually very effective at moving the design back to feasibility).

It should be noted that if only the objective function portion of the user's code is computationally expensive (or only the constraint portion) there is no need to construct meta-models for both parts of the problem: **OPTIONS** will happily work in mixed mode. To achieve this it is merely necessary to ensure that the data-base variable containing the data for the part that is not needed is emptied before invoking response surface modelling. This is achieved with the command "Modify_Data_Record OPT_TRACE Clear" (or on OPT_CTRACE for clearing constraint data) from the main command menu or using a fortran call of the form "CALL SETVV('OPT_TRACE',-1,-1,-1,0.0,ERROR)" when using the API interface (or the equivalent C call to setvv_ if working with C in a unix environment).

Six example macro command files called "bump.rsm_clf", "bump.rsm_rbf", "bump.rsm_ply", "bump.rsm_shep", "bump.rsm_sp" and "bump.rsm_ei" are included in the standard installation "cmds" subdirectory that demonstrate the use of response surface methods on the 2d version of the bump test problem. "bump.rsm_shep" uses the basic Shepard model, "bump.rsm_ply" uses polynomial regression, "bump.rsm_rbf" uses the Radial Basis Function model, "bump.rsm_sp" uses the stochastic process model for the objective function and "bump.rsm_ei" additionally uses the expected improvement operator on the objective. These macros illustrate the methods when modelling both objective and constraints together and separately. They also show the use of space filling searches to fill the data arrays to begin with. Plots are produced which show the base problem and the modelled functions both with the original data and after updating following a series of searches. These can be compared to the search of the full problem. These examples make clear the difficulty of dealing with a hard constraint boundary using a meta-model: the searches often end on the constraint boundary and this is often not precisely accurate, so that when the full model is invoked the design moves from feasibility into infeasible space. Thus the search ends up exploring the constraint boundary and response surface models are not good at describing such features. "bump.rsm_clf" illustrates the use of the **OPTIONS** search engine itself to tune the hyper-parameters in the stochastic process model as these can sometimes be difficult to set with a simple one stage simplex search, which is the default process.

# CHAPTER 11

## Multi-Objective Optimization

Many design search and optimization problems do not have a clear single goal: often the designer is trying to satisfy a number of conflicting desires and requirements. The normal approach to such problems is to try and decide on the most important goal and set this as the objective function and then to cast all other requirements and goals in the form of constraints. For example if low cost *and* low weight are desired, say, it may be possible to set weight as the objective and then to set an upper limit on cost as a constraint or vice versa. Alternatively, it may be possible to formulate a hybrid objective such as the weighted sum of cost and weight. Sometimes, however, none of these tactics suffice and a more overt approach to multi-objective optimization is required. Formally, if a problem has multiple, conflicting, objectives and these cannot be combined then the problem leads to the construction of a Pareto front or surface and the idea of Pareto Optimization[22].

This chapter of the manual outlines the basic ideas of Pareto optimality and describes the **OPTIONS** Pareto optimization system.

### 11.1. Pareto Fronts

A Pareto front is formed from a set of design solutions to a single design problem where each member of the set is optimal for at least one goal. What this means is that in moving from any single design in the set to any other, although one or more objectives in the problem may be improved, at least one other is made worse. Formally, a design is said to be Pareto optimal if it is not dominated by any other design in the set where to dominate means that at least one objective is better and all others are no worse. A related and slightly simpler way to visualize this process is to consider a set of designs such that for each member of the set it is always possible to chose weights ($w_i$) in the following equation such that any design has a better aggregate objective than all the others :

$$aggregate \ \ objective = \sum_{i=1}^{n} w_i objective_i$$

(this assumes, of course, that all the objectives should either be maximized or minimized and is called an extremal minimum set - it is not quite identical to the Pareto set, however, and will miss some designs that are strictly Pareto optimal - the extremal minimum set finds application when all designs in a set are Pareto optimal and some further form of discrimination is desired). Given a complete Pareto front for the problem it is usually a trivial exercise to select the overall best design if the relative importance of each goal is fixed. Thus trade-off exercises simply involve moving along the Pareto front and the goal of the optimization process can be seen as the construction of a complete front. This can, however, be quite difficult and is always more time consuming that finding an individual optimal design. Moreover, when dealing with problems with very many objectives (more than 20, say) it is possible to find that all designs in a reasonably large sample are Pareto optimal. In these circumstances the previous equation may be applied for a series of weights in an attempt to discriminate further - this must be carried out manually, however, as **OPTIONS** does not currently offer this facility.

### 11.2. OPTIONS and Pareto Fronts

To use **OPTIONS** to construct Pareto fronts it is necessary to specify the problem in a somewhat formalized fashion within the constraints of a traditional single objective optimization problem. Then, any of the normal methods within **OPTIONS** can be applied to construct the front, although only a rather small subset of the full suite of methods usually proves effective in carrying out this task.

Setting up the user's problem is quite straightforward as the more complex joint objective function part of the problem is pre-defined for the user in the file "parfun.f" (which provides the routine OPTFUN for **OPTIONS**). This routine sets the formal objective of the search to be the construction of a Pareto front with a good coverage of the user's multiple goals (which must be defined as minima, i.e., if any of the goals must be maximized then their negations must be used as targets). The user's multiple goals are instead placed within the user defined constraint routine OPTCON (optcon_ in C), alongside any problem specific constraints.

Within OPTCON each goal is set by reference to an **OPTIONS** variable of the form "PARETO_OBJ(1,1,II)" where II runs from one up to the total number of goals in the problem (at most 99). This variable is specified within the **OPTIONS** data-base as a constraint along with a limit array "U_PARETO_O" having the same dimensions. The limit array should be filled with relatively easily achievable target values for the corresponding objectives at the start of the search (i.e., the starting point should satisfy all these constraints and certainly they should all be higher than the final desired goals). The optimizers then seek to produce a Pareto front that provides a wide range of solutions while ensuring that all goals exceed their initial targets (setting higher, and therefore more demanding, initial targets for some goals via the U_PARETO_O limit array can be used as a mechanism to steer the search into particular directions, although the basic objective function in parfun.f aims to give even coverage).

Having set the problem up in this way any of the search mechanisms within **OPTIONS** can then be used to construct a Pareto front which is stored in the variable PARETO_FNT. Previous experience shows that the evolution strategy optimizer OPTES usually performs very well on this task[23]. This whole process is best illustrated by example, see the following sections.

### 11.3. Example Pareto Optimization

By way of example, consider again the beam design problem described in Chapters 4 and 5, but now assume that alongside minimum cross-sectional area an additional goal is to minimize the height to breadth ratio for the cross-section. Clearly, since the optimal design found in the earlier chapters is limited by the height to breadth ratio constraint it is likely that reductions in this ratio will lead to greater cross-sectional areas: i.e., the two goals are in conflict and so a Pareto front can be produced. The code used for the routine OPTCON is based on merging those for OPTFUN and OPTCON described in Chapter 4 and listed in Appendix E and is as follows:

```
      SUBROUTINE OPTCON(ERROR)
C
C        beam model for multi-objcetive optimization
C
C-----------------------------------------------------------------------
C
C        Routine specific type declaration block
C
      LOGICAL ERROR,DEFLT
      DOUBLE PRECISION GETVVD,BRDTH,HEIGHT,FORCE,LENGTH,EE,GG,NU,FACTOR
     1 ,SIGMAB,TAU,DEFLN,FCRIT,HONB,SIGMAY,AREA
      REAL GETDEF
C
C-----------------------------------------------------------------------
C
C        The main areas of routine code now follow.
C
C        n.b. breadth in mm and height in cm
C
      ERROR=.FALSE.
      BRDTH=MAX(1.0D-3,GETVVD('BREADTH',1,1,1,ERROR))
      IF(ERROR)GOTO 999
      HEIGHT=MAX(1.0D-3,10.0D0*GETVVD('HEIGHT',1,1,1,ERROR))
      IF(ERROR)GOTO 999
C
C        cross-sectional area
C
      AREA=BRDTH*HEIGHT
```

```
C
C       save results
C
      CALL SETVVD('AREA',1,1,1,AREA,ERROR)
      IF(ERROR)GOTO 999
C
C       set parameters
C
      LENGTH=MAX(1.0E-3,GETDEF('LENGTH',1,1,1,1500.0,DEFLT))
      FORCE=GETDEF('FORCE',1,1,1,5000.0,DEFLT)
      FACTOR=GETDEF('FACTOR',1,1,1,2.0,DEFLT)
C
C       decide material type and set material properties
C
      EE=GETDEF('EE',1,1,1,216620.0,DEFLT)
      GG=GETDEF('GG',1,1,1,86650.0,DEFLT)
      NU=GETDEF('NU',1,1,1,0.27,DEFLT)
      SIGMAY=GETDEF('SIGMAY',1,1,1,200.0,DEFLT)
C
C       set material limits (tau-y = sigma-y / 2)
C
      CALL SETVVD('U_SIGMA-B',1,1,1,SIGMAY,ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('U_TAU',1,1,1,SIGMAY/2.0D0,ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('L_F-CRIT',1,1,1,FORCE*FACTOR,ERROR)
      IF(ERROR)GOTO 999
C
C       bending stress
C
      SIGMAB=6.0*FORCE*LENGTH/(BRDTH*HEIGHT*HEIGHT)
C
C       shear stress
C
      TAU=1.5*FORCE/(BRDTH*HEIGHT)
C
C       tip deflection
C
      DEFLN=4.0*FORCE*LENGTH**3/(EE*BRDTH*HEIGHT**3)
C
C       twist buckling instability load
C
      FCRIT=(4.0*SQRT(GG*(BRDTH**3*HEIGHT+HEIGHT**3
     1 *BRDTH)*EE*(BRDTH**3*HEIGHT)
     2 /(144.0*(1.0-NU*NU)))/(LENGTH*LENGTH))
C
C      section aspect ratio
C
      HONB=HEIGHT/BRDTH
C
C       save results, limiting them to sensible numbers
C
      CALL SETVVD('SIGMA-B',1,1,1,MIN(1.0D15,SIGMAB),ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('TAU',1,1,1,MIN(1.0D15,TAU),ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('DEFLN',1,1,1,MIN(1.0D15,DEFLN),ERROR)
```

```
      IF(ERROR)GOTO 999
      CALL SETVVD('F-CRIT',1,1,1,MIN(1.0D15,FCRIT),ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('H-ON-B',1,1,1,MIN(1.0D15,HONB),ERROR)
      IF(ERROR)GOTO 999
C
C        save multiple goals in Pareto Objective array
C
      CALL SETVVD('PARETO_OBJ',1,1,1,MIN(1.0D15,AREA),ERROR)
      CALL SETVVD('PARETO_OBJ',1,1,2,MIN(1.0D15,HONB),ERROR)
C     CALL SETVVD('PARETO_OBJ',1,1,2,MIN(1.0D15,DEFLN),ERROR)
      IF(ERROR)GOTO 999
C
C-----------------------------------------------------------------------
C
C        The remaining lines are standard routine exit code.
C
  999 RETURN
      END
```

As can be seen from this code, apart from concatenating the code from the original two routines together all that has been added is the storing of the multiple goals into the array PARETO_OBJ. This routine is then used alongside the routine "parfun.f" to create a problem specific program in the normal way with the main **OPTIONS** library (under UNIX using something like "f77 -o options main.o opt.f parfun.f beam3con.f -lgendat" followed by any other graphics or maths libraries depending on the specific details of the installation - a site specific "make file" is normally supplied containing such information).

### 11.4. Modifying the Data-base

Having built a running program the previously defined data-base must be slightly modified. In particular four new variables must be added (PARETO_OBJ of type Constraint and dimensions 1,1,2; U_PARETO_O of type Limit and dimensions 1,1,2; PARETO_FNT of type Parameter and dimensions 2,$n$, $m$ and PARETO_QTY of type Obj_Funct and dimensions 1,1,1). PARETO_FNT is where **OPTIONS** stores the evolving Pareto front and its dimensions are set to 2,$n$,$m$ where $n$ is either the dimension of the current problem vector (i.e., how many designer chosen variables there are) *or* the number of goals currently being studied, whichever is larger while $m$ is the maximum number of designs to be kept as the Pareto front set. The elements 1,$i$,$j$ refer to the $i$th design variable in the $j$th element of the Pareto front while the elements 2,$i$,$j$ refer to the $i$th design goal value in the $j$th element of the Pareto front. These dimensions are automatically controlled by **OPTIONS** and the user need only create the array with dimensions 1,1,1. PARETO_QTY is where the routine "parfun.f" returns its current assessment of the quality of the Pareto front and this replaces the objective function previously used (AREA which should be set to type Dependant). The macro command file "beam3.dbs" makes these changes to "beam1.d" if this file has been created previously (using "beam1.dbs") and results in the creation of "beam3.d".

At the end of a search the Pareto front may be studied by the user to decide which design best meets current requirements. The graphics style file "pareto_front" may be used to plot this front if it consists of no more than three design goals. Figure 11.1 is generated by the use of this style file at the end of the macro command file "beam3.dbs" using the OPTES optimizer (Objective 1 is the area and Objective 2 the ratio of height/breadth for the beam design problem of Chapters 4 to 6).

### 11.5. Response Surface Models and Pareto Fronts

It is possible to combine the stochastic process modelling capabilities of **OPTIONS** with the construction of a Pareto front when only two objectives are being studied. To do this a SPM is built for each goal, one in the objective function space and one in the first constraint space. This is a slightly different approach to the previous scheme and relies on **OPTIONS** to construct a combined expected improvement function from the two models as a single function which can then be used in optimization. The theory behind this approach is described by Keane[24]. Within **OPTIONS** this method in fact involves fewer changes to the user's problem set up then the direct search for Pareto fronts already described. No changes are required to the user's objective or constraint function codes or data-base provided the second goal function is set up as the first of the constraints and that the variables described in the previous sections have been added to the data-base (the ordering of constraints is simply controlled using the "Modify_Data_Record" command to set the record number of the desired constraint to be lower than that of all others). To search for the Pareto front the process is then first seeded by running a design of experiments search to place data in the OPT_TRACE and OPT_CTRACE arrays from which the intial SPM surfaces can be constructed, see Chapter 10 and the macro command file "beam4.dbs" which makes the desired changes to "beam1.d" (see Chapter 4) creating "beam4.d".

Having created a suitable data set the dual SPM's are invoked by turning on RSM with codes 4.41 to 4.46 for the objective (the constraint RSM for the first constraint is invoked automatically, and the user's full constraint code is used alongside the SPM of the first constraint). Here code 4.41 gives a model based on the combined probability of improvement, including augmentations to the front, 4.42 is combined probability of improvement but requiring all new points to dominate a member of the old front, 4.43 uses an average between these two measures while 4.44 to 4.46 are similar but based on the expected improvement rather than the probability of improvement. In general the higher numbers are more aggressive searches but find it less easy to deal with deceptive problems. Notice that this will also cause any other constraints to be modelled by polynomial RSM's. If the user prefers the remaining constraints to be instead evaluated using the problem specific code supplied in OPTCON, then having built and evaluated the joint RSM once using the 4.4x series codes, the user simply renames the OPT_CTRACE trace array to hide it from the RSM system on subsequent calls. The joint RSM tool allows for this since once the joint RSM is built it is preserved for subsequent work until over-written. To over-write it (for example when updates are to be added) the array is simply renamed back to OPT_CTRACE and the function re-evaluated.

The resulting combined response surface based model replaces the original objective function with a SPM RSM for the joint probabilty / expected improvement of the Pareto front. This can then be searched using any of the **OPTIONS** methods and an update (or updates) found that can then be evaluated using the user's full code and added to the trace arrays (see Chapter 10 for further details of the best ways of doing this). Typically more updates should be used than the number of points in the initial DoE. All constraints are applied as per the single objective problem by direct calls to the user's constraint code (e.g., beam1con.f). The macro command file "beam4.run" uses "beam4.d" to do this creating 50 updates (this is a slow process taking around one hour since careful hyperparameter tuning is used).

At the end of a search the Pareto front may be studied by the user to decide which design best meets current requirements. The graphics style file "pareto_front" may be used to plot this front if it consists of no more than three design goals. Figure 11.2 is generated by the use of this style file at the end of the macro command file "beam4.run" using the GA optimizer to carry out updates (in this case Objective 1 is the area and Objective 2 the bending stress for the beam design problem of Chapters 4 to 6). Figure 11.3 shows the points on the front plotted against the design variables (as $\times$) and also all the other points from the original DoE and the full update sequence (as +).

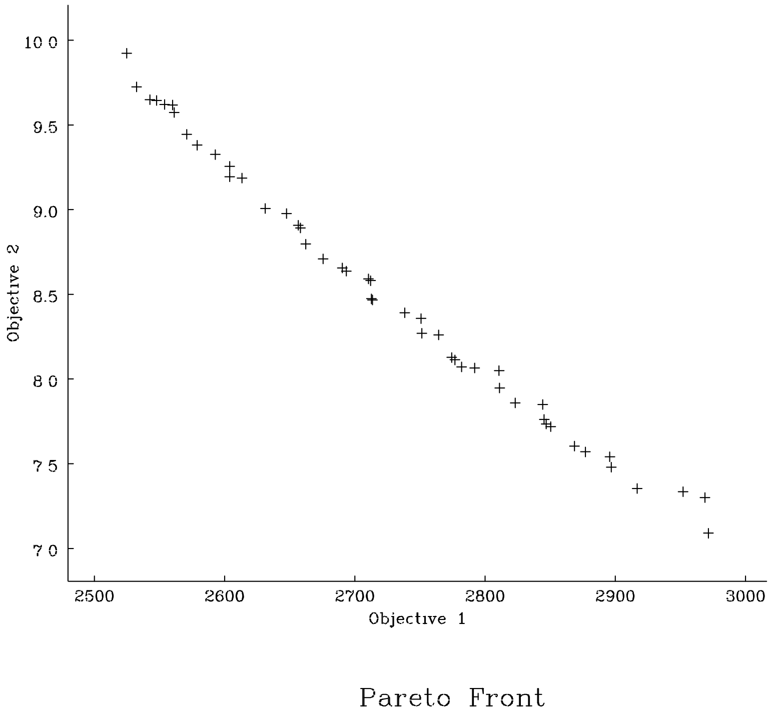



Figure 11.1 - Pareto front of area versus height/breadth ratio.

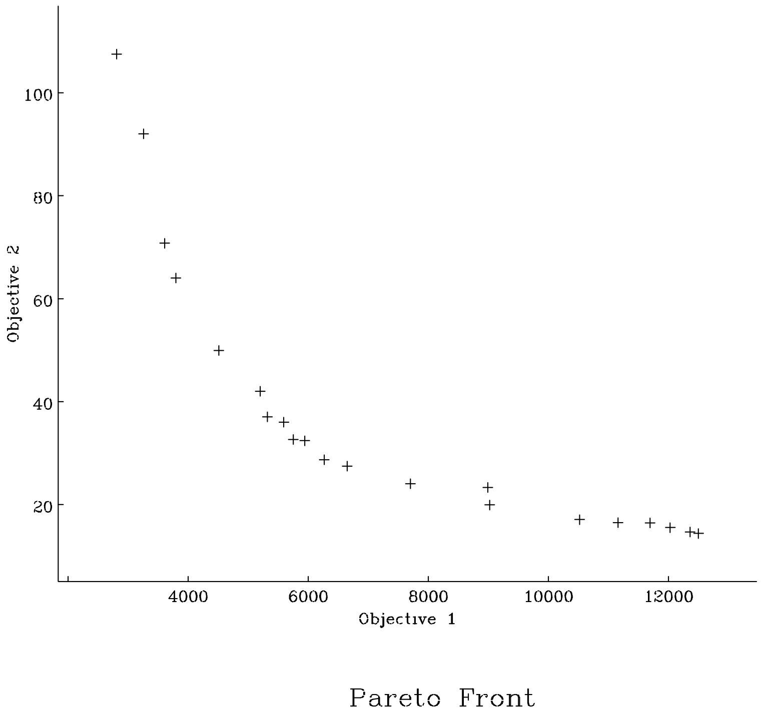



Figure 11.2 - E[i] based Pareto front of area versus bending stress.

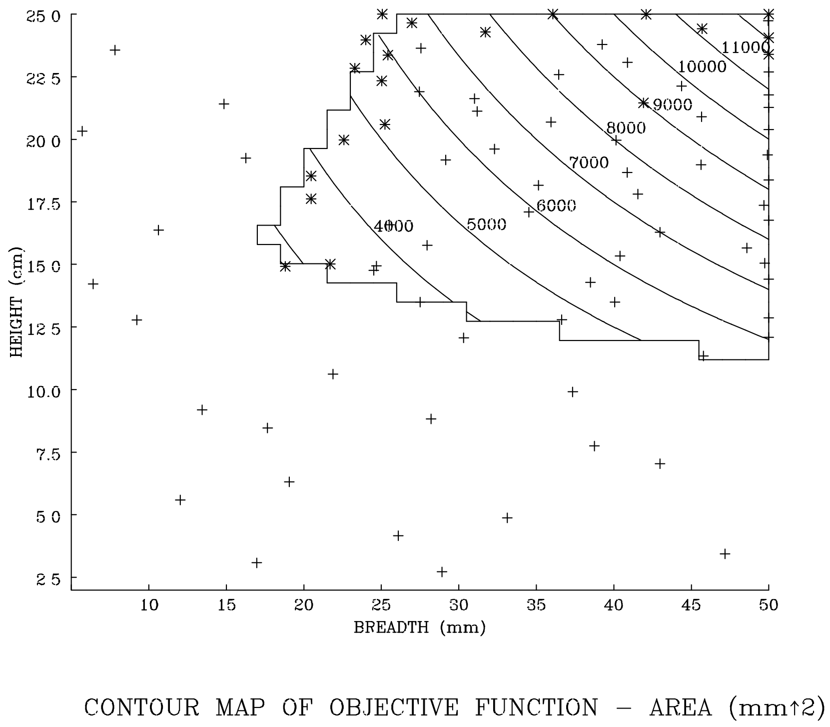



Figure 11.3 - E[i] based Pareto front of area versus bending stress (also showing DoE and update points).

# CHAPTER 12

## Summary

This manual has described a design exploration system which draws on established and modern state-of-the-art optimization methods. It has been illustrated by its application to a simple beam design problem using both manual and automated searches. The main advantage of the **OPTIONS** system is its ability to contrast and compare a variety of approaches to design, using both manual and automatic methods for the synthesis *and* analysis of competing designs.

The manual design example demonstrates how the various facilities within **OPTIONS** can be applied with full control by the designer without having to worry about the interfacing between modules. This feature arises from the adoption of a central design data-base which keeps track of the current state of the design, preventing calculations from being performed when insufficient information is available, etc.

The automated example given illustrates that optimization based concept exploration is fully capable of identifying the best design, given a suitable objective, constraints and a set of variables to manipulate. However, it also reveals that the use of optimizers without some care can give rise to problems, they being like any other specialized tools in this respect. The development of the **OPTIONS** system in this area continues, aiming to embed optimization skills within an expert system as part of an overall package, enabling designers, unskilled in the use of optimizers, to have better access to the technique.

The choice of a suitable objective to measure quality in design, whether made by the user or an expert system, may be regarded as a severe limitation to any goal oriented design methods. However, such arguments apply equally well to all approaches to design. If a designer is unable to decide which of a number of competing designs is best, this probably demonstrates a lack of clarity in the original specification. Moreover, using numerical measures to evaluate competing designs forces the designer to distinguish between true objectives and simple constraints; is minimum weight the real aim or should the design merely attempt to get below some maximum weight threshold ? Conversely, is a given load capacity essential or should this quantity form part of a measure of merit, as something to be improved provided that the effects on other goals are not penal ? In fact, the overt requirement to quantify some measure of merit would, itself, seem to be a desirable pressure in moving design from the realms of art-form to that of science.

Conversely, the specification of the relevant constraints and design variables for optimization is relatively straight-forward; the optimizers quickly reveal which constraints and variables are redundant. They also identify any missing constraints by their tendency to move into obviously absurd areas when fundamental limits are not applied. For example, if strength is ignored and minimum weight selected as the objective in the beam example studied here, the optimizers quickly drive the dimensions towards zero ! Indeed, the ability of optimization techniques to afford a measure of tutorial assistance to new designers would appear to be one of the great advantages of such methods.

Finally, it is worth noting that a manual such as this cannot hope to give the same coverage as the very many books devoted to the subject of design optimization. The user is therefore strongly advised to read the literature cited in the bibliography, particularly the books by Siddall[2] and Goldberg[6].

# APPENDIX A

## GENDAT User Guide

### A.1. Introduction

This user guide briefly describes the use of the GENeral purpose DATa handler package, GENDAT, when it is incorporated into a larger program. The aim of the package is to allow the user to manipulate data within a program in a highly flexible way that is consistent from program to program. When the package is added to a program it handles all data input, output and display functions and additionally provides a command processor that may be tailored to the specific program. The command language and facilities available are described in the on-line help file available to the package - just type help (see notes below).

Of course all programs that make use of the package will carry out functions other than just data manipulation. So that these functions may be merged neatly into those supplied by GENDAT, the package allows the program writer to add a sequence of commands to those intrinsic to the package. The user of the program is then presented with a core of up to sixteen GENDAT functions plus the program specific ones. The merged commands will usually vary as the user proceeds through the main program so that he may be guided into making the correct choices in the right order. At each point these might be as limited as a simple "Continue" option through to as many as twelve commands each with their own arguments and subcommands.

Program data is handled by the package in the form of a number of RECORDS. Each record consists of a variable NAME and TYPE together with three array SUBSCRIPTS, a UNITS string and a MEANING string. As such, a record forms the basic element of data and each one usually corresponds to a single (scalar, vector or matrix) variable in the user's problem. Whole records may then be displayed, added, deleted and so on. It is the variable names that the main program uses to identify each record when attempting to retrieve data from or send data to the package, therefore the main program options will often take a variable name as an argument, telling the program to carry out work on the specified record. Notice that each record may refer to a single number, in which case the three array subscripts will all be one or perhaps even omitted, or to an array of numbers, with up to three dimensions. By convention these are given in the order X, Y, Z and are limited to maxima of 99, 99 and 9,999 respectively.

When a program which uses the package is run, an initialisation phase takes place during which a data-file *may* be automatically read, which will then define the data structure to be worked from. If this is so the predefinition may be as limited as specifying the available variable types, or it may specify both types and records and may even be as detailed as to assign numerical values to some or all of the records in the structure. Thus when the user is first given the choice of an option it is wise to use the "Type" and "List" commands to examine the state of the structure. It is most likely that at this stage the user will have to add some problem specific variables and/or associated numeric values before proceeding. If a file is not read on package initialisation, the user will need to completely define a data structure before proceeding, perhaps by asking for a file to be read using the "Recall_Data" option, or by constructing one from scratch using the "Set_Data_Types" and "Add_Data_Record" options.

If the user attempts to proceed without having made available the variables or values needed within the specific program, the package will detect this and give an error message. The calling program will then usually allow the user to manipulate the structure before proceeding by adding missing variables or inputing missing numeric values. Alternatively it may abort the chosen command to allow the user to try another, or to try again. One option at this point is to store the data structure, abort the program and start again.

**Notes**

(1) The package will accept commands in lower or upper-case formats.

(2) Command words do not need to be entered in full, provided sufficient of the leading characters are specified to avoid ambiguity. Variable NAMES should always be given in full and upper-case, although this latter requirement is not strictly necessary.

(3) The help facility may be used to give information on a specific command by typing "help <command>".

(4) Entering a "!" will usually initiate error correction or terminate the operation of a command, whilst a "?" will cause the system to indicate what sort of input is expected. When learning to use the system these two conventions should be used frequently to aid familiarity. When the "?" is used the type of input required may be a keyword, in which case a list of keywords is given, or it might be a request for numerical or textural input in which case a quoted text description such as "a number" will be given. Sometimes either sort of input will work or a default will be chosen in the absence of a word.

(5) Depending on the implementation being used, a number of the GENDAT options may be suppressed to avoid confusing the inexperienced user. Entering a "?" at the option level will list those that are available, which should be sufficient to work the program. The "Full_Control on" option allows *all* the GENDAT package options to be used, whilst "Full_Control off" resets to those originally allowed. This facility should be used with care.

(6) If a sequence of commands are used repeatedly they may be saved in a macro file with one command per line, using the system editor and then invoked from the program by typing :

"!file_name n/m"

where "file_name" is the name of the file concerned and "n/m" specifies which commands within the file should be used. This range is optional and "*" may be used to indicate the first and/or last commands, respectively. Specifying a single number gives just the one command. Lines beginning with a "#" are treated as comments and ignored, those beginning with a "$" are used for special, macro file, control commands (n.b. comments and special commands *are* included in the count of lines when using sub-ranges of files). These features allow for interactive command files to be written that can take different courses during processing depending on the user's input. The valid special control commands are:

$echo which just echos the remainder of the line to the screen (note that the first character after the "$echo" is ignored on the assumption that it is a blank);

$pause which just pauses processing of the command file until the "return" key is hit;

$resp which allows the user one response before command file processing continues, if the response is not valid another is allowed until a valid one is given;

$user which gives the user control until the string "$ready" is entered whereupon command file processing resumes;

$silent which suppresses all normal GENDAT output during processing of the file (fatal error messages are not affected) and

$resume which resumes normal GENDAT output to the screen.

If command file processing results in an error processing is halted with an error message indicating where the fault happened. This is a fail safe position to stop unruly command files causing damage. It is recommended that command files that allow user input always follow that input with a lone "!" on three consecutive lines to force the GENDAT package into a known condition to allow for wrong responses by the user !!.

(7) If the XView interface is available it may be invoked by starting the overall program with the single argument of "MENU_ON". Button pushes with the left mouse button will then select commands from the available menus. A comma or space signifies the completion of a typed input where this is required. A carriage return or mouse hit on the "Complete_with_Defaults" button will initiate processing of the desired command, adding defaults to complete the command if needed. Normal keyboard interaction may always be used when using the XView interface by clicking the left hand mouse button in the lower, display window (the upper menus may be reactivated by clicking on them with the left mouse button). The lower display window is the only place where macro

command files may be initiated when using this interface, since the upper area does not support commands other than those available within the normal menus (n.b., when processing command files all the relevant menus will appear in sequence, which can slow down the processing of long command files: in such cases run the files without the XView interface).

**Intrinsic Commands**

As was noted earlier, GENDAT provides upto sixteen intrinsic commands. These are described in the following sub-sections. Notice that all commands are given in the format :-

Option, Argument, Sub-option

(argument and sub-option are often optional). The processor incorporates automatic pagination where required. Within the processor, unambiguous leading substrings are accepted. Numbers and commands should be separated by commas or spaces, except where a prompt for "n/m/l" etc. is given, which requires slashes.

Help

Usage - This option gives help on the other options.
Arguments - The valid main option word on which help is required.
Sub-options - The amount of help required "Brief or Full".
This option accesses the in built help facility to give the user simple instructions for using the program. If help is requested on "Help" a general synopsis is given, followed by help on all the available commands and finally a summary of the commands.

Quit_to_System

Usage - This option terminates execution of the program.
Arguments - none.
Sub-options - none.
This option stops the program and returns the user to the computer operating system. Before carrying out the termination it asks if the user is sure he wishes to stop and it returns to the option level if he is not. N.B. all unstored data is lost when the program is stopped, use the "Store_Data" option if you do not wish to loose it.

Full_Control

Usage - This option allows all the possible options to be used.
Arguments - "On" or "Off".
Sub-options - none.
This option allows experienced users access to the full power of the package but should used with care since it allows subtle and significant changes to be made to the program data structure. BEWARE !!

Store_Data

Usage - This option stores the current data in a file of given name.
Arguments - The valid system file name.
Sub-options - none.
This option writes a file containing the complete data structure, in a form that may only be simply read by the program. If the given file name exists it checks that the user is happy to overwrite before proceeding. It stores sufficient data to completely reinstate the data memory. The default file name is the one last used in a successful file access or if none has been performed, "gendat.d", "gdata" or "/usr/local/lib/gdata". N.b., command arguments are limited to 25 characters in length and so if a longer file name is needed the user should specify a "*" for the name and then when prompted for a new name simply hit return, at which point a long file name may be used. Note, however, that a truncated form of this name will be stored within the internal data-base and made the default file name for future file access comands.

Recall_Data

Usage - This option recalls a set of data from the file of given name.
Arguments - The valid system file name.
Sub-options - none.
This option reads a file previously created with the "Store_Data" option back into the program. It

resets all variables, their contents and types. The default file name is the one last used in a successful file access or if none has been performed, "gendat.d", "gdata" or "/usr/local/lib/gdata". If an error occurs during the file read the data arrays are completely reset to the empty condition. N.b., command arguments are limited to 25 characters in length and so if a longer file name is needed the user should specify a "*" for the name and then when prompted for a new name simply hit return, at which point a long file name may be used. Note, however, that a truncated form of this name will be stored within the internal data-base and made the default file name for future file access comands.

List_Variable_Names

Usage - This option lists the names of currently defined variables.
Arguments - The valid variable type or "*" for all types.
Sub-options - The additional information required with the list of names.
This option lists the names of variables of the given type (or all types) and optionally appends their meanings or array dimensions. Alternatively it gives those that have no data values stored in them or those that have data values but whose values have not been used.

Display_Data_Types

Usage - This option displays the currently defined types.
Arguments - none.
Sub-options - none.
This option allows the user to display the variable types that are currently available for use. It does not allow existing types to be deleted or new ones added.

Type_Data

Usage - This option types data to the screen in various formats.
Arguments - The variable name, "*" or "n/m" (where n or m may be "*").
Sub-options - The quantity of information or element members to be displayed.
This option displays the range of data desired with the degree of information requested. The variable name may be replaced with "*" for all variables or "n/m" for variables with record numbers in the range given. If a range is being given the "*" is taken to mean first or last respectively. Sub-option "Full" gives all the data on a variable or variables, "Short" omits
the meaning string and is the default, "Value" gives brief value, name and units information, whilst "Matrix" outputs items or arrays of up to two dimensions in matrix format of up to ten cols and rows. If the array is bigger than this the user is prompted for the upper left and lower right element addresses. Specifying a sub-option in the form "n/m/l" gives just the elements desired and allows large arrays to be probed. Any of the subscripts may be replaced by "*" to mean all values of that subscript. If the argument is "Data_Usage" the data usage text is displayed instead.

Print_Data

Usage - This option prints data or creates a print file.
Arguments - valid variable name "*" or "n/m" (where n or m may be "*").
Sub-options - The element members to be printed.
This option is used to print data via the terminal auxiliary port if one exists or alternatively to create a user named file that may be sent for printing after program termination by system commands. The variable name may be replaced with "*" for all variables or "n/m" for variables with record numbers in the range given. If a range is being given the "*" is taken to mean first or last respectively. Specifying a sub-option in the form "n/m/l" gives just the elements desired and allows large arrays to be probed. Any of the subscripts may be replaced by "*" to mean all values of that subscript. The data usage text and name are always printed with the other details. If a print file has been created during a program run subsequent prints may be appended to it, or put in new files.

Input_Data_Value

Usage - This option inserts numeric values into the data structure.
Arguments - The variable into which the data is to go or "Data_Usage".
Sub-options - "Full" a numeric value, "n/m/l" or "ri/rs".
This option is used to input a numeric value into a previously defined data record. If "Full" is given then all elements of the array are requested in turn (to skip between array levels the "!" is used). When inserting single values, if the variable given is not an array then the numeric value is inserted directly, whilst if it is then the "n/m/l" format is used to define the element of interest. If two real numbers (with decimal points) are given as the sub-option this is taken to indicate an increment and

starting value so that a whole array may be filled. Thus if a single real number is given to any type of variable all elements assume that value (which has the same effect as giving an increment of zero). If no sub-option is specified the user is prompted for the element to be defined, unless the variable is not an array, when only the single item may be input. If the argument is "Data_Usage" the data usage text is requested, subsequent to a check before over-writing any existing text. This text, of up to ten lines allows a note on the contents of a data structure to be held together with the data and may be used for any purpose. It is always prepended to output produced with the "Print_Data" option. This is the default argument.

Modify_Data_Record

Usage - This option modifies the characteristics of existing variables.
Arguments - The variable name, "*" or "n/m" (where n or m may be "*").
Sub-options - The characteristic to be modified.

This option is used to modify the characteristics of a previously defined data record. The variable name may be replaced with "*" for all variables or "n/m" for variables with record numbers in the range given. If a range is being given the "*" is taken to mean first or last respectively. "Clear" resets the record to a blank condition with no data and the set and get counters at zero; whilst "Kill" simply removes the record completely. The remaining sub-options require the user to respecify the parameters concerned. Inputing a "?" in name or type change mode gives a list of the currently defined names and types. A name change causes the numeric data to be deleted and the set and get counters to be reset.

Add_Data_Record

Usage - This option adds a new variable record.
Arguments - none.
Sub-options - none.

This option is used to add a new variable data record to the data structure. Note that it does not input a value to the record defined. This is achieved using the "Input_Data_Value" option. All the record parameters are requested in turn and the process may be aborted at any point by typing "!". Specifying a record number of zero places the new record after the last existing record.

Set_Data_Types

Usage - This option manipulates variable types.
Arguments - The data type or "*" for all types.
Sub-options - The change required on all variables of the given type.

This option allows the data variable types to be set or the parameters of entire types to be changed at one time. "Add" and "Kill" just add or delete variable types, noting that the "*" argument may not be used with the "Add" sub_option. If any variables of a given type exist when the type is deleted using the "Kill" sub-option, then they are also deleted (a check is incorporated to remind the user of this action, allowing the command to be aborted). "Values" causes all the variables of the type to be declared as unused (and thus resets the set and get counters to zero), whilst "Set_Ctrs" sets the set counters to one (if they are already non zero) and "Get_Ctrs" sets the get counters to zero.

Global_Information

Usage - This option gives global information.
Arguments - none.
Sub-options - none.

This option display the common block global information which shows how much room is available and has been used in the data storage area. It also lists the current data types, their usage, the data structure name and its usage.

Order_Data_Structure

Usage - This option re-orders the data structure.
Arguments - none.
Sub-options - none.

This option re-orders the main common data blocks to improve program execution speed. It does not alter the display ordering. The revised ordering is based on the sum of the set and get counters for each record, with the most frequently used records placed in the most easily found positions.

Pswitch

Usage - This option sets the debugging print switch.

Arguments - "List", "Off", depth of routine output or routine name.
Sub-options - "Brief" or "Full" output, "Add" or "Remove" routines.
This option allows the user to turn the debugging print switch on or off. On is indicated by a positive integer argument or the name of a routine. A number specifies the depth of subroutine nesting to which output data will be given; a name indicates that that routine will be added or removed from the list of routines for which output will be given, irrespective of the depth of nesting. The "List" argument lists those routines already added. The "Off" argument turns the facility off. If the sub-option is given as "Full" then output is given for all routines, even those in the various Fortran libraries; otherwise it is given only for problem specific routines.

Screen Dump of Typical XView Interface and Graphics Window

# APPENDIX B

## GENDAT Documentation File

**B.1. Introduction** This documentation file gives details of the GENeral purpose DATa handler and command processing package called "GENDAT". The main purpose of this package is to free the programmer from the need to write a user friendly and flexible method for entering data and instructions to every new program he produces. In the jargon of computer science it is what is often referred to as a "User Interface Manager" or UIM system.

The most important process it caters for is the entry of numerical data into programs and the storage of such data in files between runs of programs. This facility is provided by means of a problem orientated data-base which is maintained within the GENDAT routines. To use the system at this base level a programmer needs only to understand the basic workings of three routines; "GENDAT", "SETVV" and "GETVV".

**B.1.1. GENDAT** "GENDAT" is the fundamental routine that invokes the package and allows for database manipulation to take place; it is used in one of five modes. The most important of these allows for terminal interaction and supports up to sixteen intrinsic commands with optional arguments and sub-commands together with full on-line help facilities. The commands are used in a menu format with keywords, etc.; all of which may be displayed to help in gaining experience of driving the package. Using these facilities a problem specific data-base is built up to hold important data in a highly flexible way. This data is then passed to and from the problem specific routines provided by the user via the two routines "SETVV" and "GETVV".

**B.1.2. SETVV** "SETVV" (or "Set Variable Value") has only one function; to assign a numeric value to a specified data element from within a program. It is not an interactive routine. That is it retrieves a value from the user's program and passes it to the GENDAT data-base.

**B.1.3. GETVV** "GETVV" (or "Get Variable Value") has the converse effect to "SETVV" allowing a numeric value to be returned to a program. "GETVV" is is also non-interactive. That is it extracts a value from the GENDAT data-base and passes it to the user's program.

At a more sophisticated level additional commands may be added to the standard data manipulation commands built into "GENDAT". Such commands allow a tailor made set to be built up and presented to the user in a well defined and standard manner. This more advanced use requires a more detailed knowledge of the system and in particular of the routine "DEFVS" (or "Define Variable System"). At the most advanced level up to twelve additional commands, each with associated arguments and sub-commands may be added to form a new menu and this menu may be dynamically changed during program execution to direct the user. The added commands will have help information given with them and this is merged into the existing information to give a full on-line help facility.

**B.2. Typical Simple Implementation** A typical minimum implementation of the package will require the programmer to divide all his variables into two camps. Those that he requires full access to or which require storing between program runs and those which are entirely internal to his routines. This former set usually correspond to given variables or parameters within the physical problem being tackled (and may be scalar, vector or matrix in format); it is these variables that will be handled within GENDAT, with each one being allocated a RECORD or data group. This record consists of a name, array subscripts (all one for scalars, etc.), numeric value or values, units, type and meaning. These last three have no effect on the operation of the GENDAT routines but allow large sets of data to be handled in a neat and tidy way whilst reminding the user of the specific assumptions within the program. This data structure is then created by using interactive commands within "GENDAT" and is preserved for further use in a data-file. This operation is typically only carried out while a program is being written, since when it is finished all that the user will require is to insert numeric values or display them, rather than to create new elements (consequently the commands made available to the programmer are more powerful than those usually given to the user). A space is available within the data structure to allocate a name and brief description to each data structure created thus, so that the programmer or user can be reminded of where a file came from (one of the main advantages of the system is that any program is capable of reading *all* data-files, whether or not they were intended for the program in use).

Having separated the wheat from the chaff and allocated names, etc., to each variable of interest all subsequent references to these items from within a program are handled by "SETVV" and "GETVV". The package carries out checks that the relevant variables exist and have values ascribed to them, etc. To manipulate these variables calls to "GENDAT" are made at the required points in the program and the intrinsic functions then allow all the desired operations to be carried out. (Note that values are held within GENDAT in DOUBLE PRECISION accuracy and truncated when accessed by SETVV and GETVV, for full precision the equivalent routines SETVVD and GETVVD use DOUBLE PRECISION values.)

Note that the package uses the fortran logical unit number 11 for a number of purposes and therefore this LUN should be avoided in user written code.

An example program is included at the end of this file. It demonstrates most of the features discussed in the documentation.

**B.3. Advanced Programming** At a more advanced level the programmer must decide on the problem specific commands required within his program and whether or not these commands require arguments or sub-commands. A command structure is then built up where the command words, allowed responses and help information are specified to the routine "DEFVS" in a formalised way. This is inevitably rather tedious but once mastered allows for highly flexible and impressive programs to be created with ease.

When "DEFVS" is called with the appropriate information an interactive terminal session is begun which is similar to those generated by "GENDAT" except that the simple "Continue" option provided by "GENDAT" is replaced with whatever command structure the programmer adopts. This interactive session continues within "DEFVS" until one of the problem specific commands is given. The routine then returns to the calling program with the selected command, argument and sub-command detailed for appropriate action by the host program. On completion of such a command the call to "DEFVS" is usually repeated until the user finally stops the program from within "DEFVS".

To assist in such advanced uses the programmer may require program access to information other than the specific numeric values associated with given variables. To provide for this the routines "GETVN", "GETVI" and "GETGI" allow variable names, other variable information and overall global information to be returned as desired. Using these routines it is possible to recover all the information contained within a data structure.

**B.4. Batch Processing** On occasions it is useful to be able to run programs which do not require interactive sessions, merely reading data-files, carrying out calculations and then storing the results. To aid this the routine "GENBAT" allows the package to be initialised and processing carried out, using any of the set and get routines without the overheads associated with the command processor. A final call to "GENBAT" then stores the data-file before execution is stopped.

**B.5. Major Data I/O Library Modules** The DATA I/O routines held in the GENDAT subroutine library that are normally available to the user are outlined in the following sections. Further information on individual routines is given in the various files in the "man" subdirectory of the installation. In particular see the file "gendat.hdrs" which contains information on the arguments and common blocks used by the routines.

**B.5.1. GENDAT** This is the fundamental control and initialisation routine. It *must* be called at the beginning of any program that is to use the "GENDAT" package. It operates in one of five modes that allow data initialisation and command level selection to be made.

(1) The package is initialised, either to a null condition or from a specified data-file. The number of intrinsic commands to be made available to the user under normal control or by calls to "DEFVS" is set.

(2) The data structure is not changed but an interactive terminal session is begun with the option "Continue" allowing return to the main program.

(3) The data structure is completely reset, equivalent to restarting the package but without the need to actually stop the program and start again. No data-file is read and so a completely blank data structure results.

(4) The number of intrinsic commands to be made available to the user under normal control or by calls to "DEFVS" is changed.

(5) The number of intrinsic commands to be made available to the user under normal control or by calls to "DEFVS" is reset to the level previously chosen.

N.B. The last two of these allow a program with user specified sub-menus to specify different numbers of intrinsic commands within the sub-menus from that in the main menu.

Within an interactive terminal session a sophisticated command language is used to control the package. This is described in the user guide and also by the on-line help facility, just type "help" within the package.

**B.5.2. SETVV** This is a simple subroutine that allows a single REAL number to be entered into a variable (or array element of a variable). If the variable does not exist within the data system or the array subscripts given are out of limits it gives an error message and returns with an error flag set ".TRUE.". If INTEGER or COMPLEX variables are in use they must be converted to REAL prior to storing and in the case of COMPLEX variables stored in two parts (DOUBLE PRECISION variables are handled by the routine SETVVD which is entirely similar, but just works with the higher precision data). If all three subscripts are given as minus one the entire variable is cleared of all numeric data and the set and get counters reset to zero.

**B.5.3. GETVV** This is a simple REAL function that allows a single REAL number to be retrieved from a variable (or array element of a variable). If the variable does not exist within the data system, the array subscripts given are out of limits or no value has been ascribed to the element it gives an error message and returns with an error flag set ".TRUE.". If INTEGER or COMPLEX variables are in use they must be converted from REAL after recovery and in the case of COMPLEX variables formed from two parts (DOUBLE PRECISION variables are handled by the routine GETVVD which is entirely similar, but just works with the higher precision data).

**B.5.4. DEFVS** This is potentially the most powerful routine within the package. To get the best from this routine an understanding of UIMs is beneficial. A basic menu concept is used and all responses to every possible command fixed either to be keywords or specified textural elements. This allows the command processor to check the validity of all responses and thus prevent typing errors and the like. It also allows the on-line help utilities to display all possible responses at any point and also to allocate default actions.

The command typed in by the user is parsed into three command words which are separated by either blanks or commas. No more than three words are allowed in this input and the maximum word length is twenty-five characters. If a "!" or "?" is detected in the sequence special action is taken and the command either aborted or help given respectively. Assuming the input consists of three normal words these are checked against the allowed keywords or textural types. This checking happens sequentially with the first word being used to select which secondary and tertiary words are allowed. The non-keywords types allowed are single numbers, groups of integers, groups of real numbers and free-format text (since blanks and commas delimit whole words, number groups are separated by "/"s and of course the free-format input cannot contain blanks or commas). Notice that if a free-format textural string is permitted then any non-blank response is allowed. Also should a complete sentence be required then special actions must be taken to prevent the command word parsing action.

To allow for rapid input unambiguous leading sub-strings are allowed for the keywords whilst if a null input is given (only permitted if at least one keyword is specified) then the first keyword is taken as a default. In the returned command string all the keyword inputs are replaced by the complete explicit words specified within the program. This string is then displayed so that confirmation of the chosen command is given.

Up to twelve additional commands each with up to twenty-six arguments and sub-commands may be added to the inherent sixteen. Help information is added on these but this is limited to just three lines compared to the maximum of twenty-one allowed for the intrinsic commands. newrec.f

**B.5.5. GETVN, GETVI and GETGI** These subroutines extract various items from the data system for subsequent use by the calling program.

**B.5.6. SETVVD and GETVVD** DOUBLE PRECISION versions of the routines GETVV and SETVV, otherwise identical in operation.

**B.5.7. GETVVS** DOUBLE PRECISION version of the routine GETVV that accepts the specification of the desired data-base element as a CHARACTER*(*) string but otherwise identical in operation.

**B.5.8. SETDEF and GETDEF** These subroutines allow for data access when the variable used may or may not exist within the data-base, such as when debugging or providing optional input or output. GETDEF additionally allows for default values to be specified, which are used when no data-base value is available.

**B.5.9. SETDFD and GETDFD** DOUBLE PRECISION versions of the routines SETDEF and GETDEF, otherwise identical in operation.

**B.5.10. GENBAT** This subroutine simply initialises the package from a given data-file or stores the current data structure to a file.

**B.5.11. SGRAPH** This subroutine allows access to the complete Simpleplot graphics package which may then be used interactively to create, display and plot X-Y graphs, contour plots or 3-D surfaces from data held within the data structure. This facility is optional to the standard features of GENDAT and is described in separate documentation.

**B.5.12. OPTCTL, OPTDBS and OPTNS** These subroutines allow access to a large library of optimization software which may be used to specify and solve optimization problems based on the data held within the data structure. This facility is optional to the standard features of GENDAT and is described in separate documentation.

**B.5.13. Other GENDAT routines** In addition to the main routines a number of very specialised routines are also available to carry out a variety of advanced functions. Further information on individual routines is given in the "man" subdirectory of the installation. In particular see the file "gendat.hdrs" which contains information on the arguments and common blocks used by the routines. These routines should only be used by very experienced users as they can completely destroy data structures and invalidate the integrity of codes being written. The library contains:

| | |
|---|---|
| ADDREC | adds a new record to the data structure. |
| CHGDIM | changes the dimensions of a data record. |
| DATAAN | analyses the statistics of a GENDAT data array. |
| GDHELP | supplies on line help information. |
| GETSTR | carries out data-base value substitutions into strings. |
| GIGCTR | returns the get counter of record without checks. |
| GISCTR | returns the set counter of record without checks. |
| GSTACK | gets variable values in double precision via stack pointers (to a user supplied program). |
| INDVAL | inputs data variable values. |
| IOCOMM | carries out file I/O operations. |
| LVNAMS | list the variable names in the structure. |
| MODREC | modifies data records. |

MODTYP modifies data types.

NEWREC creates new records in the data-base without user I/O.

PRTDAT prints parts or all of the data structure.

SETGI sets global information on data name and usage.

SIGCTR sets the get counter of record without checks.

SISCTR sets the set counter of record without checks.

SSTACK sets variable values in double precision via stack pointers (from within a user supplied program).

TRNSFM allows for variable values to be transformed (added, multiplied, etc.,).

TYPDAT types variable values to the terminal.

USRDF2 user defined function of two variables.

USRDFN user defined function.

USRFN2 user defined function of two variables.

USRFTN user defined function.

**B.6. FORLIB** In addition to these routines a library of more general purpose routines called "FORLIB" is also available to carry out a whole variety of common functions. Further information on individual routines is given in the "man" subdirectory of the installation. In particular see the file "forlib.hdrs" which contains information on the arguments and common blocks used by the routines. The FORLIB library contains

ANSWYN poses a yes or no question and returns the answer.

ASKSYS a C routine to allow a fortran program to give a command to the system and receive a one line reply.

AUXPRT couples the auxiliary port on the terminal to allow printing.

CHKMNU checks individual words against menu lists.

CMENU a C routine to provide a window, icon, mouse, pointer or W.I.M.P. interface to menus handled by "GETMNU" using XView.

COMPAR compares two strings to see if one is a sub-string of the other.

CONECT connects a logical unit for file transfers.

CREAD a C routine that allows the UNIX standard input to be read without a hang-up if no reply is received in a certain time.

CSYS a C routine to allow the C "system" call to be made from fortran as the UNIX "f77" intrinsic version causes problems with file control.

DMTINV inverts square matrices giving the determinant in DOUBLE PRECISION.

DMTLUD forms LU decompositions of matrices.

FD3INT carries out a GILL and MILLER 3rd order FD integration.

FNDWRD compares a word to a list of words to find which one is chosen.

GET1NO gets one integer or real number from the terminal in free format.

GETMNU uses menus to return up to three command words.

GETNUM gets up to six real numbers from the terminal in free format.

INTERP a Lagrangian polynomial interpolation routine.

IODAT a common block data statement, invoked automatically.

MATINV inverts square matrices giving the determinant.

MATOUT allows matrices to be displayed in a flexible way.

NOTDIS a C routine to call the window system notifier.

| | |
|---|---|
| PACKWD | condenses extraneous blanks out of sentences. |
| PATDUP | a duplicate of "PATSON" to allow 2-dimensional integration. |
| PATSON | a numerical integrator based on Paterson rules. |
| RANDNO | generates pseudo random numbers. |
| RANLST | generates random lists of integers. |
| READNO | converts a 25 character word into a number. |
| REGRES | carries out least squares polynomial regressions. |
| RSTRNG | reads a string from the terminal or a command file (basic I/O) |
| RWRANG | converts a 25 character word into three real range numbers. |
| SIGCON | a C routine to remove UNIX interrupt (Ctrl-C) signal catching. |
| SOLVE | a routine for driving "INTERP" if a suitable problem form exists. |
| UPRCAS | converts a 25 character word to upper-case. |
| WRANGE | converts a 25 character word into three integer range numbers. |
| WSTRNG | writes a string to the terminal (basic I/O) |

Some of these functions are UNIX specific and are not implemented on other operating systems.

**B.7. Data Structure** The data structure, which is held in the COMMON blocks DAT001/2, consists of a number of data records, one per variable defined, holding all the information on each variable, a DOUBLE PRECISION data VALUE stack accessed by the POINTERs, and a number of GLOBAL ITEMs giving detail on the overall structure. The records are accessed by reference to their IDENTIFIERs or RECORD NUMBERs which are used to find the POSITION of the desired record in the COMMON block. The relevant numerical VALUE is then recovered from the data stack by reference to the associated POINTER. Therefore records may be placed in any order within the data structure to help the overall display. Furthermore if high access speeds become a requirement, it is possible to use the variable SET and GET COUNTERs to re-order the data according to usage. When this is done the display ordering is preserved since items are always displayed in RECORD NUMBER order. The data consists of the following numerical and textural elements:

### B.7.1. Name

COMMON BLOCK:- DAT002        ARRAY:- VNAME

The variable NAME is used when accessing the variable VALUEs within problem specific, user written FORTRAN code (using the routines SETVV, GETVV and GETVI) or when defining the variable structure. It consists of a CHARACTER*10 element, and is used together with three INTEGER array subscripts to form an IDENTIFIER (the array subscripts should be set to one for non-array variables, and are limited to a maximum of 99 for X and Y and 9,999 for Z). If during program execution, an attempt is made to access an un-identified variable a suitable warning is displayed and control reverts to the calling program to allow for error corrections. N.B. the NAME should be given in complete upper-case format when using the routines SETVV, GETVV and GETVI although within the command processor built into DEFVS a mixed case un-ambiguous leading substring will suffice.

### B.7.2. Array Dimensions

COMMON BLOCK:- DAT001    ARRAYS:- MXSUB, MYSUB, MZSUB

Three INTEGER elements that indicate the maximum ARRAY DIMENSIONs of the relevant variable. They are set when a given variable is defined or its dimensions modified (up to a maximum of 99 for X and Y and 9,999 for Z). For non-array variables these should all be set to one. They are used to check the array bounds on any attempt to access a given variable and also to calculate the storage requirements for the numerical VALUEs assigned to each array. Any attempt to access outside the bounds of an array during program execution results in a suitable warning being displayed and control reverts to the calling program to allow for error corrections.

### B.7.3. Type

COMMON BLOCK:- DAT002 ARRAY:- VTYPE

The variable TYPE is indicated by a CHARACTER*10 element which is used to subdivide the variable structure to facilitate block modifications. A list of variable types is maintained and only TYPEs contained within the list are permitted (see global items 2.11.). The TYPE may be of mixed case and include non-alphabetic elements (except "?", "/", "!", "*", "," and blanks).

### B.7.4. Units

COMMON BLOCK:- DAT002 ARRAY:- UNITS

A CHARACTER*6 element that is input by the user to indicate the units of the relevant variable. It is given with the numerical VALUE in display or print options. The UNITS may be of mixed case and include non-alphabetic elements (except "?", "!" "," and blanks).

### B.7.5. Meaning

COMMON BLOCK:- DAT002 ARRAY:- MEANG

A CHARACTER*60 element that is input by the user to describe the use or meaning of the relevant variable. It is output when full data display or printing is required. This string may contain blanks and will usually be in the form of a single sentence. The MEANG may be of mixed case and include non-alphabetic elements (except "?" and "!").

### B.7.6. Record Number

COMMON BLOCK:- DAT001 ARRAY:- IRECNO

An INTEGER element that indicates the position, in the output, that a given element should occupy for display and printing options. It is set when a given variable is defined or its definition is modified. If it is given as a number that is already used a re-ordering of the RECORD NUMBERs is carried out. When adding new records selecting a RECORD NUMBER of zero positions the new record after the last existing record.

### B.7.7. Value

COMMON BLOCK:- DAT001 ARRAY:- VALUES

The DOUBLE PRECISION numerical VALUE associated with the variable, which is used during problem analysis. These items are held in a stack structure with the location being indicated by the associated POINTER (see below). If INTEGER, COMPLEX or LOGICAL values are required then these must be gained by processing the available DOUBLE PRECISION values (by using NINT, etc.). REAL values are provided automatically by the routines SETVV and GETVV.

### B.7.8. Set Counter

COMMON BLOCK:- DAT001 ARRAY:- ISCTR

An INTEGER element that is incremented by one each time the relevant variable is set during program execution. At the beginning of each run all the SET COUNTERs for those variables to be calculated during the run, should be reset to zero, which allows subsequent analysis of the frequency of accessing each item. If, during program execution, an attempt is made to use the VALUE of an unset variable, a suitable warning is displayed and control reverts to the calling program to allow for error corrections. At the end of a run any variable that has not been set may be pointed out to the user.

N.B. the resetting of counters prior to runs, and display of unused variables after runs is accomplished through the options "Set_Data_Types" and "List_Variable_Names" within the routine DEFVS, respectively.

### B.7.9. Get Counter

COMMON BLOCK:- DAT001        ARRAY:- IGCTR

An INTEGER element that is incremented by one each time the relevant variable is used during program execution. At the beginning of each run all GET COUNTERs should be reset to zero, which allows subsequent analysis of the frequency of accessing each item. At the end of a run any variable that has not been used may be pointed out to the user with a suitable message.

N.B. the resetting of counters prior to runs, and display of unused variables after runs is accomplished through the options "Set_Data_Types" and "List_Variable_Names" within the routine DEFVS, respectively, (note further that the GET COUNTER can only be non-zero if the SET COUNTER is also non-zero).

### B.7.10. Pointer

COMMON BLOCK:- DAT001        ARRAY:- POINTR

An INTEGER element that indicates where in the stack of data VALUEs the relevant item occurs, or in the case of arrays, the first item in the array. Arrays are stored with the Z subscript varying most quickly and the X subscript least.

### B.7.11. Global Items

COMMON BLOCK:- DAT001/2

The following GLOBAL ITEMs are held in the data structure to help in storage and access routines (all INTEGERs except for TYPES, DATNAM and DATUSE):

MVARS Indicates the maximum total number of variable records that may be defined in this program implementation, currently 200.

NVARS Indicates the total number of variable records currently defined.

MVALS Indicates the maximum total number of variable VALUEs that may be used in this program implementation (limited by MVARS if no array variables are to be used), currently 100,000.

NVALS Indicates the total number of variable VALUEs currently defined.

MTYPS Indicates the maximum total number of variable TYPEs that may be defined in this program implementation, currently 20.

NTYPS An array, running from a subscript value of zero up to MTYPS, that indicates the total number of variable TYPEs that are defined in the current structure (as the zeroth element) and the number of records of each TYPE (as the subsequent elements).

TYPES An array of up to MTYPS character variables holding the currently defined variable TYPE names in CHARACTER*10 format.

DATNAM
A CHARACTER*25 variable that holds the name of the data structure being worked on. This is use for default file access and is appended to printed output. It is updated whenever a file storage or recall action, with a specified name, is carried out. The package warns the user if a file being recalled has differing file and data structure names.

DATUSE
A CHARACTER*60 array of ten elements that holds ten lines of data structure description. This element allows the user to add comments or notes to a data structure so that if the structure is separated from the program that created it, its purpose is apparent. This is input in free format by the user with the "Input_Data_Value" command used with no arguments. The text is added to the top of any printed out and is displayed if the "Type_Data" command is given with the argument

"Data_Usage".

N.B. When a TYPE, RECORD or VALUE is deleted from the structure a repacking sequence is carried out so that there are never any gaps in the structure. Then when items are added they may be merely placed at the end of the structure until the total number is equal to the maximum handled by the given implementation.

**B.8. Logical Unit Numbers** Logical unit numbers within the package are specified by the original call to "GENDAT" or by default values in the block data routine "IODAT". They may be accessed within the main program by including a suitable common block if desired. It is of the form:

```
LOGICAL IOCHCK
INTEGER INLUN,OUTLUN
COMMON/IOLUN/INLUN,OUTLUN,IOCHCK
SAVE /IOLUN/
EXTERNAL IODAT
```

**B.9. Terminal Input Facilities** To provide a consistent interface four of the routines in the "FORLIB" library are used throughout the package in place of normal terminal input via fortran "READ" statements. These routines are customized to suit the data required and allow for yes or no questions ("ANSWYN"), a single REAL or INTEGER number ("GET1NO"), a group of up to nine REAL or INTEGER numbers ("GETNUM") or an arbitrary CHARACTER*(*) string ("RSTRNG"). It is strongly recommended that these routines are used to replace all terminal input in the main program.

**B.10. Terminal Output Control Facility** All the routines accessed by this package incorporate a general purpose output control facility. If the package needs to produce output this is done by making writes to an internal character array instead of a fortran logical unit number directly. This internal write is then followed by a call to the routine "WSTRNG" which then produces output instead. This routine incorporates facilities to concatenate repreated messages and to suspend output during command file operation to save time and simplify use. It may be incorporated into the main program if desired by adding a suitable common block. It is of the form:

```
CHARACTER*256 OUTSTR(50)
COMMON/CWSTRG/OUTSTR
SAVE /CWSTRG/
EXTERNAL IODAT
```

*All* write and print statements should then be changed to internal writes to this block followed by calls to "WSTRNG" with two INTEGER arguments; the first being the desired logical unit number for output and the second a code between zero and three indicating the importance of the message (three being the highest priority and zero normal priority).

**B.11. Debugging Facility** All the routines accessed by this package incorporate a general purpose debugging print switch, which may be incorporated into the main program if desired. It is of the form:

```
LOGICAL PRTSW, PFULL, PMUST
INTEGER LOGLUN, LEVEL, PLEVEL, PCOUNT
COMMON/CPRTSW/PRTSW, PFULL, LOGLUN, LEVEL, PLEVEL, PCOUNT
SAVE /CPRTSW/
CHARACTER*6 PNAMES(10)
COMMON/CPRTNM/PNAMES
SAVE /CPRTNM/
EXTERNAL IODAT
```

Here LOGLUN is the logical unit for debugging output (which is set to be the same as that for terminal output, by default in the initialisation block data statement). If output to a file is desired this can be set up with the FORLIB routine "CONECT" and LOGLUN changed to suit. PRTSW is the logical flag used to turn the facility on or off (default is off i.e. .FALSE.) and LEVEL sets the level of output required. PFULL is set .TRUE. if full debugging output is required and this should *only* be used by user written programs to remove them from simple traces when the user is *sure* that the given routine is functioning correctly. PLEVEL maintains a count of the current level of subroutine nesting and so should be incremented on entry and decremented on exit from *all* routines irrespective of whether they will produce debugging output or not. If the name of the routine is found in the list held in PNAMES then output should also be turned on for that routine. A typical implementation, here for the routine "ANSWYN" is as follows

```
      PLEVEL=PLEVEL+1
      IF(PRTSW.AND.PFULL)THEN
      PMUST=.FALSE.
      DO 99999 PCOUNT=1,10
      IF(PNAMES(PCOUNT).EQ.'ANSWYN')PMUST=.TRUE.
99999 CONTINUE
      IF(PMUST.OR.LEVEL.GE.PLEVEL-1)THEN
      IF(PMUST.OR.LEVEL.GE.PLEVEL)THEN
      WRITE(OUTSTR,9990)
 9990 FORMAT(1X,75('-'))
      CALL WSTRNG(LOGLUN,0)
      ENDIF
      WRITE(OUTSTR,9991)
 9991 FORMAT(1X,'Routine ANSWYN - Incoming')
      CALL WSTRNG(LOGLUN,0)
      IF(PMUST.OR.LEVEL.GE.PLEVEL)THEN
      WRITE(OUTSTR,9992)PROMPT(1:LEN(PROMPT))
 9992 FORMAT(1X,'Input arguments :-'/
     1       1X,'PROMPT = ',A)
      CALL WSTRNG(LOGLUN,0)
      WRITE(OUTSTR,9993)
 9993 FORMAT(1X,'Input/Output arguments :-'/
     1       1X,'none')
      CALL WSTRNG(LOGLUN,0)
      WRITE(OUTSTR,9990)
      CALL WSTRNG(LOGLUN,0)
      ENDIF
      ENDIF
      ENDIF
```

Notice that this uses the terminal output control facility.

**B.12. Example Program** The following example demonstrates a relatively basic use of the package:

```
      PROGRAM EXAMPL
      LOGICAL ENDLST,ERR
      CHARACTER*60 HLPUSE(1),HLPARG(1),HLPSCM(1),DUSE(10)
      CHARACTER*25 CMDWRD(1),ARGWRD(1,1),SCMWRD(1,1)
     1 ,COMMND,ARGMNT,SUBCMD,DNAME
      CHARACTER*10 NAME,TYP
      CHARACTER C
      DATA CMDWRD,ARGWRD,SCMWRD,HLPUSE,HLPARG,HLPSCM/
     1 'Return',' ',' ',
     2 'Return to example program.',
     3 'none',
     4 'none'/
C
C       Decide command level
C
 50   CALL GET1NO('Input GENDAT command level.'
     1 ,'The level selected must lie between -7 and +7.'
     2 ,REALNO,LEV,ERR)
      IF(ERR)GOTO 50
C
C       Initialise package, no predefined structure file
```

```
C        available so use '*'
C
      CALL GENDAT(5,6,LEV,'*',' ')
C
C        now use the 'Continue' option to access DEFVS
C        to create the structure
C
      CALL GENDAT(5,6,LEV,'Continue','Select Option')
C
C        Fill an array with data
C
      DO 100 II=1,5
      DO 70  JJ=1,3
      RNUM=FLOAT(II)**2.3+SQRT(FLOAT(JJ)/2.0)
      CALL SETVV('VAR_1',II,2,JJ,RNUM,ERR)
      IF(ERR)GOTO 110
   70 CONTINUE
  100 CONTINUE
C
C        example of use of DEFVS with the added command "RETURN"
C
  110 CALL DEFVS('Select Option','Example use of DEFVS.'
     1 ,HLPUSE,HLPARG,HLPSCM,1,CMDWRD,1,ARGWRD,1,SCMWRD
     1 ,ICNUM,COMMND,ICNUM,ARGMNT,ICNUM,SUBCMD)
C
C       example of use of GETVV
C
      RETVAL=GETVV('VAR_1',1,2,1,ERR)
      IF(ERR)GOTO 110
      PRINT *,'RETVAL = ',RETVAL
C
C        Call GENDAT again using the continue facility
C
 150  CALL GENDAT(5,6,0,'Continue','First Continue Option')
C
C        no reset the option level
C
      CALL GENDAT(5,6,0,'*',' ')
      CALL GENDAT(5,6,0,'Continue','Second Continue Option')
C
C        now get names and set counters of all the variables of
C        type "NEWVAR"
C
      DO 200 II=1,1000
      CALL GETVN('NEWVAR',II,NAME,ENDLST,ERR)
      IF(ERR)GOTO 150
      IF(ENDLST)GOTO 300
      CALL GETVI(NAME,I,I,I,I,ISCTR,I,I,C,C,C,C,ERR)
      PRINT *,'Name = ',NAME,' Set Counter = ',ISCTR
 200  CONTINUE
C
C        now get the global information
C
 300  CALL GETGI(1,DNAME,DUSE,I,I,I,I,I,NTYP,TYP,I,ENDLST)
      PRINT *,'There were ',NTYP,' variable types.'
      DO 400 II=1,NTYP
      CALL GETGI(II,DNAME,DUSE,I,I,I,I,I,I,TYP,NTYPN,ENDLST)
```

```
      IF(ENDLST)GOTO 500
      PRINT *,'Type no ',II,' - name ',TYP,' - used '
     1 ,NTYPN,' times.'
400   CONTINUE
500   PRINT *,'Thats all folks !'
      STOP
      END
```

Note that this program requires the variable "VAR_1" to be a pre-defined array of at least 5x2x3 elements of some undefined type. If the structure is not created by the call to GENDAT then the first call to SETVV will result in an error message, but will only require the array size to be 1x2x1. The direct call to DEFVS illustrates its use but note that for a single command it is much more clumsy than an equivalent call to GENDAT. The final call to GENDAT allows the structure and results to be examined, printed and stored for future use as required. Using the "Continue" option that the final call to GENDAT adds to DEFVS will cause the package to return to the calling program. GETVI and GETVN are then used to extract variable information from the common block, whilst GETGI gets global information.

# APPENDIX C

## Graphics User Guide

### C.1. Introduction

This user guide briefly describes the use of the general purpose graphics package available with the GENDAT data-base software. This package is based on the Simpleplot library of graphics routines and plots graphs from data held within the GENDAT package. It also allows arrays of data to be manipulated in a variety of ways before plotting or for other purposes. This guide does cover all that can be done with the package, but deals with the production of simple graphs from data arrays. The package is however capable of greater things, such as producing contour and isometric surface plots, see the examples at the end of the guide. Throughout this guide instructions that must be typed are given inside double inverted commas thus "graph", this may include spaces that should be entered as shown.

There are a few golden rules to be observed when using the package :-

(1) All GENDAT based programs have an on line help facility, just type "help" or if help is needed on a particular command type "help command". If a command is very complex typing "help command full" will usually give even more detailed help on the command. Please read all the help information displayed when using these commands, do not jump to the conclusions that all the information present can be gleaned from the first sentence.

(2) Always use lower case letters for everything unless specifically asked to do otherwise. Usually the package does not distinguish between cases but an exception is the name of a file where upper and lower cases are treated differently.

(3) When giving commands to the package it is not necessary to type the entire command etc., just enough letters to distinguish what is wanted from any other possibility. That is "Recall_Data myname" which recalls data from the file called "myname" may be shortened to "re myname" (but not to "Re_Dat myname" or to "re my").

When stuck *do not* try pressing keys and buttons at random, it is unlikely that the right key will be hit by accident. When stuck "^C" will abort the package but will loose any data that has not been previously saved in a file.

### C.2. Data and Style

The graphics package divides information about graphs into two parts. These are data and style. The data is the information that is to be displayed and typically consists of a number of (X,Y) co-ordinate pairs. The style governs how the graph will look. That is the types and ranges of the axes, the shapes of symbols at points and the style of lines used. These two groups of information may be stored in files so that they may be saved before leaving the package and logging out. Remember that if an hour is spent typing in information and the package left without without telling it to save the data in files it will simply forget the information and waste the hours work !

#### C.2.1. Data

The data to be plotted must of course be typed into the package or computed internally. It is then held within the GENDAT software for subsequent plotting. There are a number of commands within GENDAT to allow data to be input, altered, displayed in tabular form, saveed to a file or retrieved from a file. The most useful ones are :-

(1) "Input_Data_Value" - which allows data to be re-input from the keyboard.

(2) "Type_Data" - which displays the input data (n.b., there are sub-options which give differing formats for the output).

(3) "Store_Data" - which stores the data in a file (n.b., if the file already exists the package will check before overwriting it).

(4) "Recall_Data" - which recalls the data from a file.

For further help on these and any other commands mentioned in this guide use the on-line help facility, type "help command", or for commands dealing with the manipulation of *data* consult the GENDAT user guide.

### C.2.2. Style

The package will plot data using default settings if other style options are not specified. These defaults include the assumption that output is to a postscript file (unless the current terminal type is defined as an X-terminal by an environment variable, in which case that is used instead). If another device is to be used this must be indicated, as producing graphics commands unsuitable for the device in use can have dramatic side effects! The default style also assumes that only one curve is to be plotted and that it consists of twenty-five points held in the GENDAT variables "XDATA" and "YDATA". The package will ignore trailing null points, however (i.e., those whose Y values are exactly zero). There are commands to allow the selection of options, their display, saving them to or retrieving them from a file. These are not part of the default GENDAT command structure, but features of the graphics interface. The most useful commands are :-

(1) "Select_Device" - which selects the output device. Notice that the selected device is *not* saved in the style file and must be selected at least once each time the package is run if the default device is not wanted.

(2) "Draw" - which draws a complete graph to the current device (includes axes, all active curves and key box if specified). Notice that none of the "Draw" commands affects the style file describing the graph.

(3) "Draw 3/5" - which draws curves 3 to 5 to the current device.

(4) "Draw Key_Box" - which draws a key box containing entries for all currently active curves to the current device.

(5) "Draw Axes" - which draws a fresh set of axes to the current device or, if axes are already drawn and the plot is still open, adds a set of axes to the right hand side of the plot.

(6) "Picture_Control New_Graph" - which declares that the current graph is finished and thus a subsequent draw will produce a new picture.

(7) "Picture_Control End_Group" - which declares that the current group of graphs has been completed and therefore closes the device, flushing any remaining commands. It is vital that this command is given before leaving the package if hard copy is being produced, otherwise the graphs will not be complete or may not be usable at all. This command should also be used if you wish to change graphics devices, perhaps because a plot to the screen has been produced and hard copy is now required.

(8) "Key_Position T L" - which turns on the key box and places it in the top left corner of the graph.

(9) "X-axis_set Linear" - which sets auto-scaled, linear X-axes.

(10) "X-axis_set Linear -5./20." - which sets linear X-axes from minus five to twenty.

(11) "Y-axis_set Logarithmic 10/1.e3" - which sets logarithmic Y-axes with two cycles from ten to a thousand.

(12) "Captions Primary" - which enables a main caption to be specified (n.b., using "Secondary", "X-axis" or "Y-axis" instead of "Primary" allows the other captions to be specified).

(13) "File_Options myname.style" - which files all the style option details in a file called "myname.style", checking if the file already exists.

(14) "File_Options myname.style Recall" - which resets all the option details from a file called "myname.style". Notice that this single command encompasses both storage and retrieval of the options, whereas there are two separate commands for the data-file.

(15) "Define_Curve" - which allows the options for specific curves to be altered. This option causes a whole new sub-menu to come into play and return to the main graphics menu is then achieved by using the "Graph_Menu" command.

The define curve sub-menu allows details for each curve to be set up. The curves being worked on are fixed by a command word appended to the various modification commands. This word may be a number (to chose a specific curve), "all" or, by default, "First_Curve". The most important commands in this menu are :-

(1) "Graph_Menu" - which just returns to the main graph menu.

(2) "Plot_Control" - which allows up to ten curves to be turned on and off.

(3) "X-data_Name" - which sets the name of the GENDAT variable to be used for the X coordinates (substitute "Y" for Y coordinates).

(4) "Range_of_Data" - which allows the specific array elements within a GENDAT record to be specified for plotting. This allows various elements from multi-dimensional arrays to be plotted against each other, etc.

(5) "Curve_Type Tight_Curve all" - which sets all curves to be tight fitting spline curves (possible types are just points, tight splines, loose splines, straight lines, polynomial regression, histograms, cumulative histograms, probability density functions or probability distribution functions - in the last four cases only the Y data is used).

(6) "Line_Type 4 8" - which sets the line type to be 4 on curve 8 (n.b., there are fifteen line types from -7 to +7 with 0 being unbroken plain lines).

(7) "Symbol_Type 3" - which sets the symbols used at the data points to be of type 3 on the first curve (by default) (n.b., there are sixteen symbol types from 1 to 16, with 16 being no symbol, moreover, symbols need not occur at each point and additionally a number may be written next to the symbol, which is useful for referring to specific points within a figure).

(8) "Pen_Type 2" - which sets the pen to be type 2 for the first curve(by default) (n.b., pens 1 to 4 are black, red, blue and green on devices that support colours).

(9) "Caption" - which allows the key box caption for the first curve (by default) to be specified.

There are other commands to allow further refinements but these should be enough for most purposes: again use the help facility.

### C.3. Transform_Data Menu

The Transform_Data command within the graphics package selects a sub-menu that contains commands for manipulating data held within the GENDAT data-base. These commands allow arrays of data to be combined, to be scaled, logarithms to be taken, etc. They also allow portions of arrays to be copied and inserted into other arrays so that sub-arrays can be plotted, etc. The most often used of these are the "Scale_and_Shift" command which allows quadratic or linear scaling plus an offset to be added to data and the "Copy_Sub_Array" command which allows parts of arrays to be copied. This latter command allows increments to be specified for each array subscript on both the source and destination arrays so that columns can be turned into rows, every other element extracted or over written, etc., allowing great flexibility in the handling of data.

### C.4. Graphics Examples

To give some insight into the range of graphics options available with the package, two GENDAT macro command files ("test.gr" and "test.3d") are supplied along with three data files ("graph.d", "cont.d" and "grid.d") and eight style files ("graph[1-6]", "cont" and "grid") which together produce two example figures each containing six plots. These twelve plots are given on separate pages here as Figures C.1-12. The first six demonstrate a number of axes settings, curve types, line styles, etc., plus the use of the "Transform_Data" menu commands, while the second six demonstrate the plotting of three-dimensional data (n.b., the use of greek letters in the key of the sixth plot and also the automatic insertion of data values drawn from the data-base; also, the gaps in the three-dimensional plots are produced using a ‘gap’ value of -1.0E20 which, when gaps are enabled using the "Contour_Control Gaps On" command, is used to mark out points in grids which are not to be contoured). Also supplied with the package is a tutorial command file which takes the user through the construction of a simple graph with various prompts and hints. This is invoked by typing "!cmds/tutor" at the first prompt.

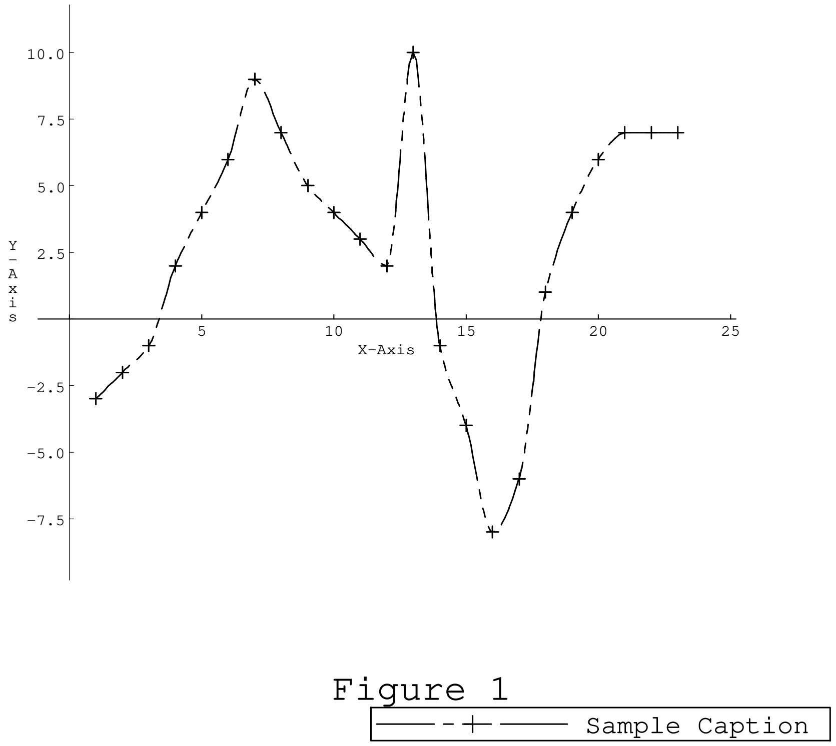



Figure C.1 - Graphics example 1(a).

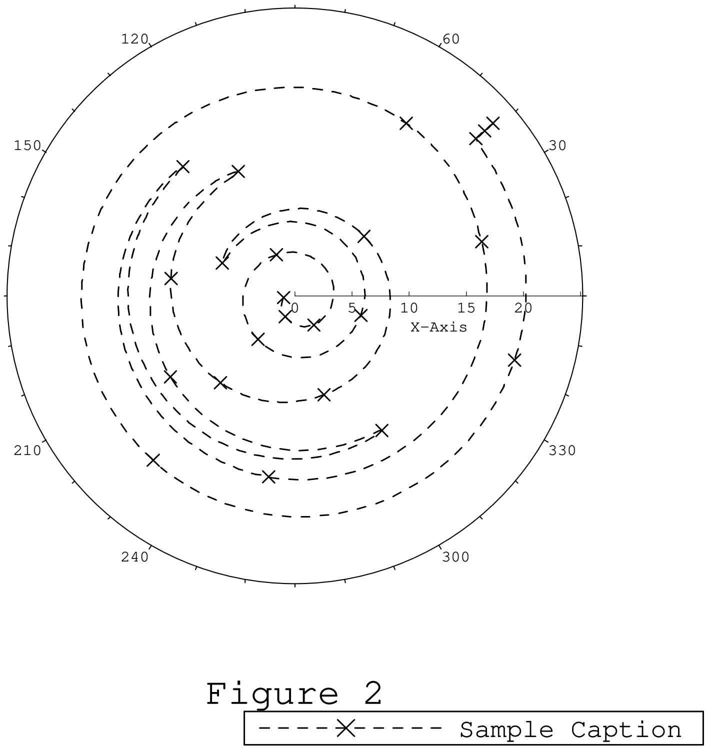



Figure C.2 - Graphics example 1(b).

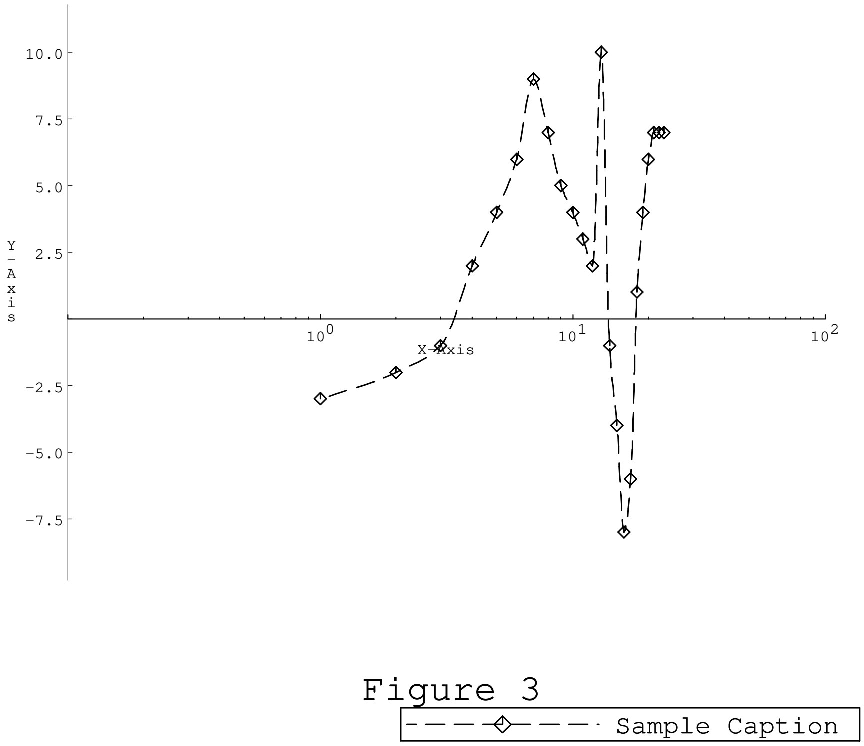



Figure C.3 - Graphics example 1(c).

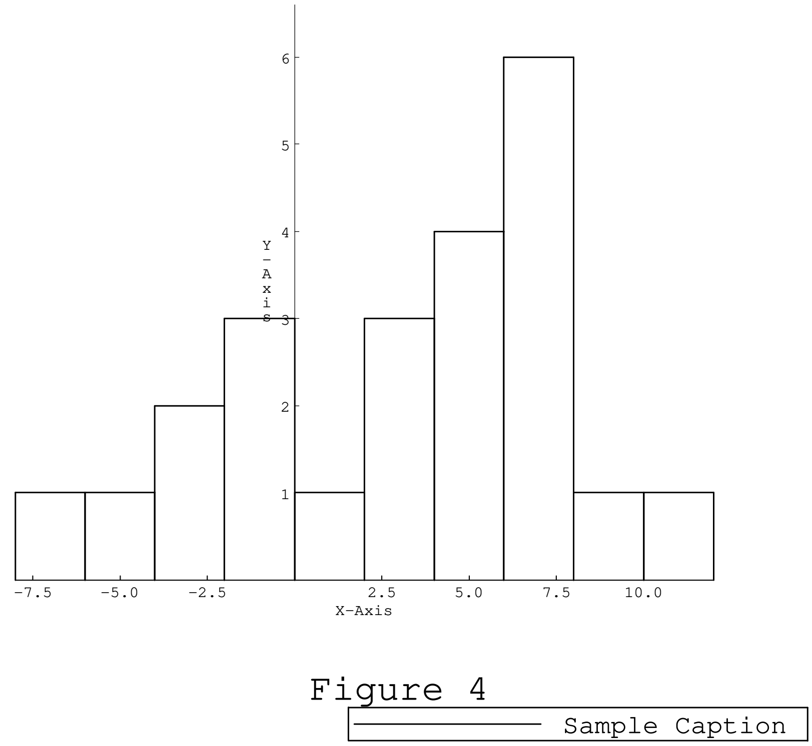



Figure C.4 - Graphics example 1(d).

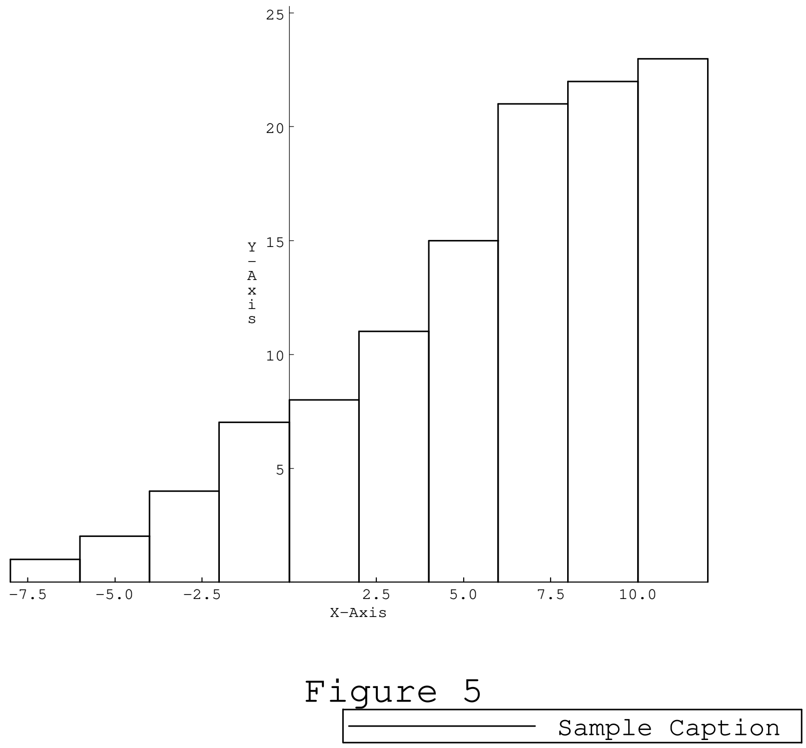



Figure C.5 - Graphics example 1(e).

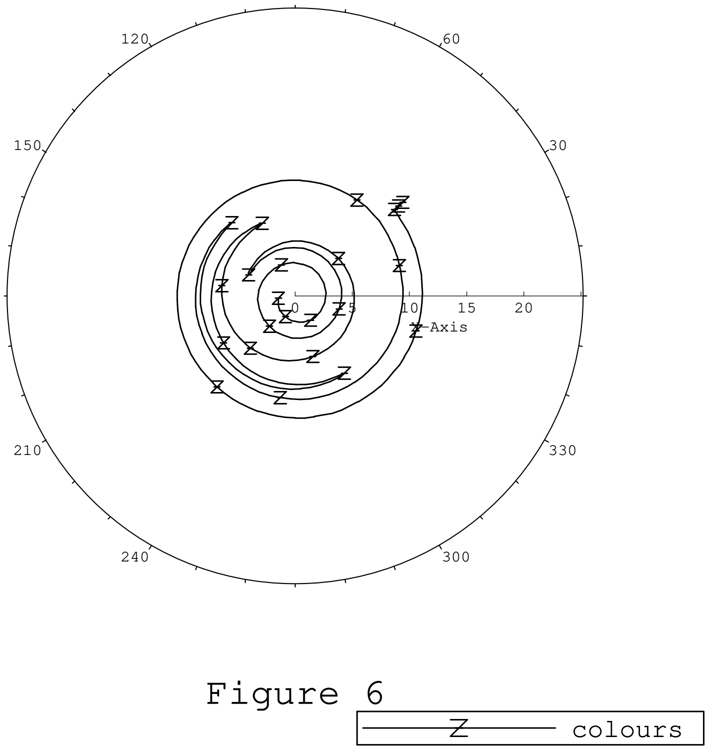



Figure C.6 - Graphics example 1(f).

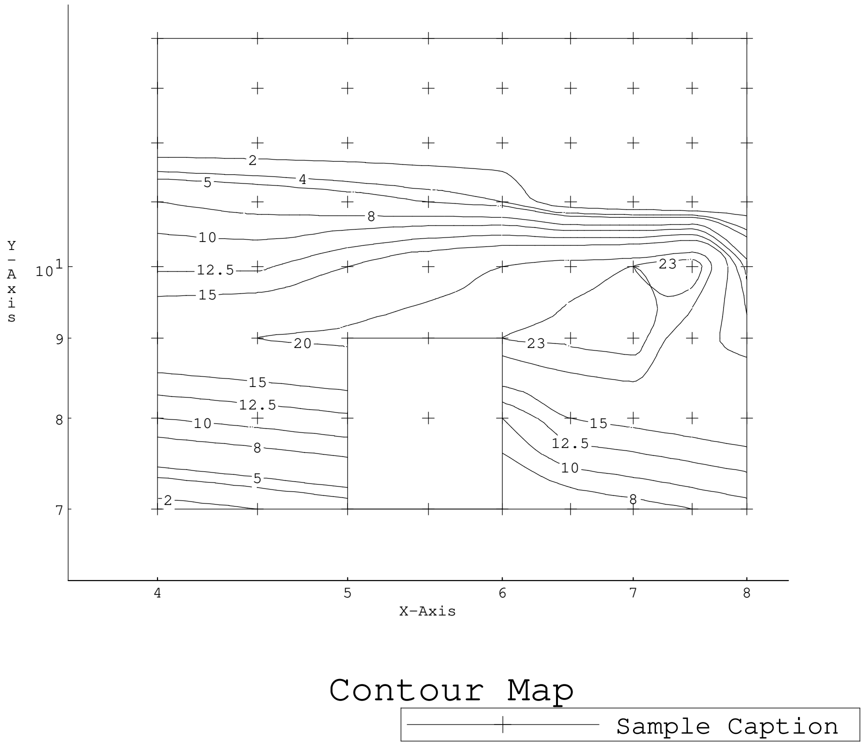



Figure C.7 - Graphics example 2(a).

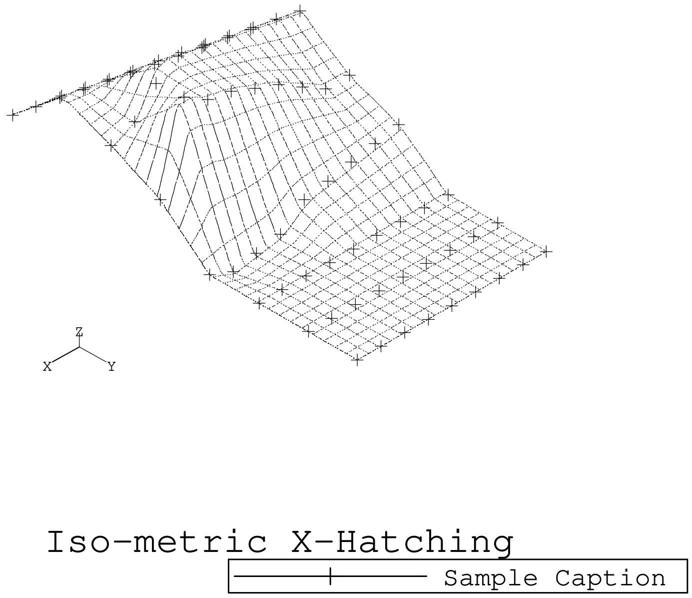



Figure C.8 - Graphics example 2(b).

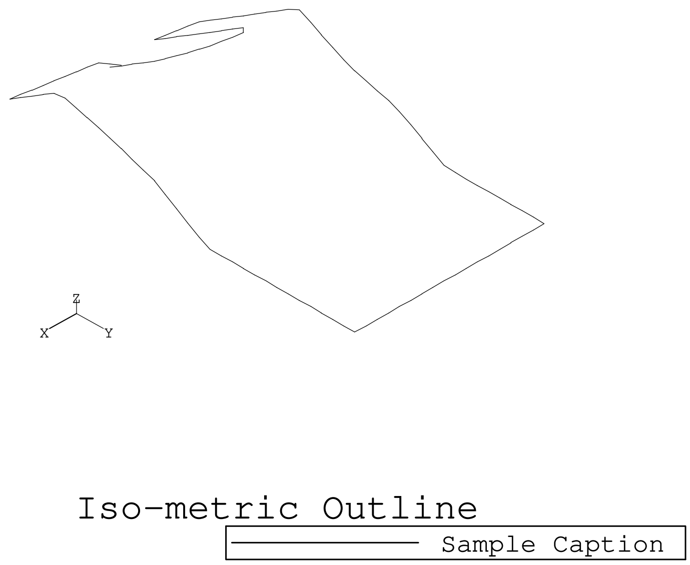



Figure C.9 - Graphics example 2(c).

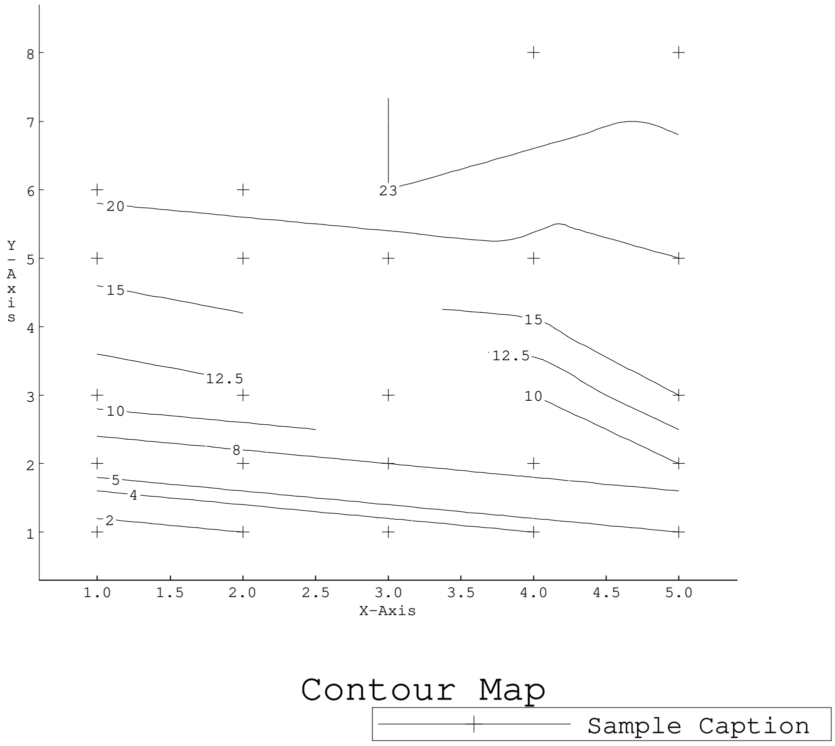



Figure C.10 - Graphics example 2(d).

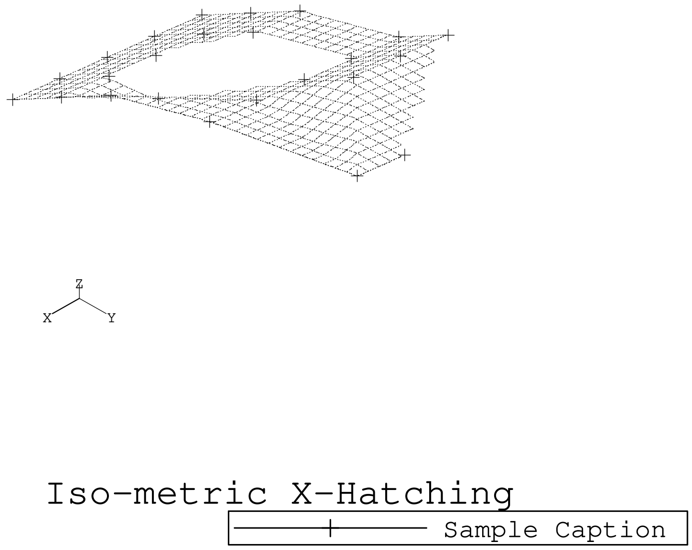



Figure C.11 - Graphics example 2(e).

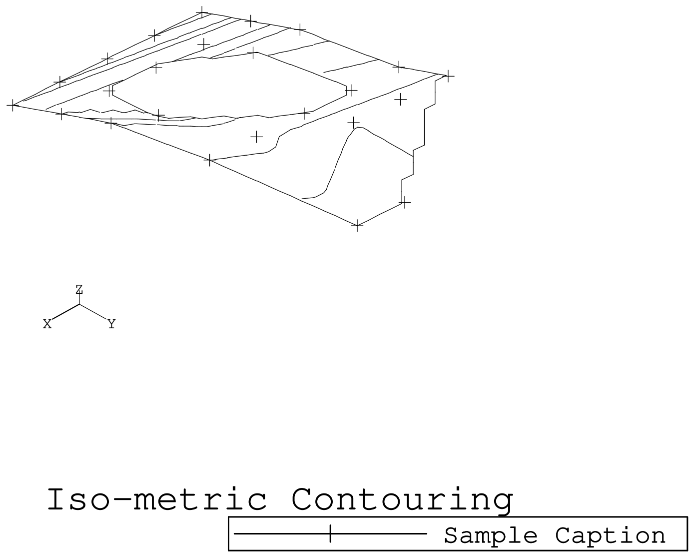



Figure C.12 - Graphics example 2(f).

# APPENDIX D

## Optimization Methods

### D.1. Introduction

This appendix outlines the theory behind the various optimization techniques available within the **OPTIONS** package. It is divided into separate sections dealing with the stochastic methods, a miscellaneous collection plus those taken from the OPTIVAR and Schwefel libraries.

All optimization methods may be classified as: a) constrained nonlinear methods; b) unconstrained nonlinear methods combined with penalty functions; c) single-variable minimization or d) linear-programming. In engineering design the general problem to be solved may be described in mathematical terms as follows:

find

$$U = U(x_1, x_2, \cdots, x_n) = \text{minimum}$$

subject to

$$\psi_i(x_1, x_2, \cdots, x_n) = 0 \qquad i = 1, m$$

and

$$\phi_j(x_1, x_2, \cdots, x_n) \geq 0 \qquad j = 1, p$$

where $U$ is the objective function, $x_1, \cdots, x_n$ are the $n$ variables that may be changed by the optimizer (the trial vector), $\psi_i (i = 1, m)$ are $m$ equality constraints and $\phi_j (j = 1, p)$ are $p$ inequality constraints (n.b., stochastic searches cannot deal with equality constraints and should not be used when these arise). Maximization is achieved by minimizing $-U$; constraints applying at values other than zero are transformed by the addition or subtraction of a suitable constant, with those where the constraint requires the function to be *below* a value replaced by $-\phi$ (these transformations are automatically invoked by the **OPTIONS** package). This problem definition means that, in general, it is constrained nonlinear methods and unconstrained nonlinear methods combined with penalty functions that are relevant to design exploration.

### D.2. Penalty Functions

A penalty function is a simple technique used to distort the optimization function in order to force the search towards feasibility, when a constraint is, or is about to be, violated. This enables any technique that does not explicitly deal with constraints to handle constrained problems. The simplest form of penalty function is:

$$U_p = U(x_1, x_2, \cdots, x_n) + r \sum_{i=1}^{m} |\psi_i| + r \sum_{j=1}^{p} | < \phi_j > |$$

where $U_p$ is the penalized objective function which is minimized instead of $U$, $r$ is a large number and the symbol $< \alpha >$ has the meaning

$$< \alpha > = \begin{cases} \alpha & \text{if} \quad \alpha \leq 0 \\ 0 & \text{if} \quad \alpha > 0 \end{cases}$$

This penalty function is used as the one pass external function with $r = 10^{20}$. This is very severe and sometimes the search stalls, particularly if it must follow along the constraint line to reach the optimum. Despite this disadvantage, the strategy is often quite successful, and requires a minimum of computer time when it works.

The other penalty functions in the suite soften this severe distortion and to try to achieve less risk of premature stalling of the search. The one pass penalty function is only active when constraints are violated and the search is in an infeasible region. Functions having this characteristic are called external penalty functions. Conversely, internal functions are active only in the feasible region, so that the surface is distorted when the inequality constraint line is approached, and the search is "warned" that there is trouble ahead. The Fiacco-McCormick function combines these approaches and takes the form:

$$U_p = U(x_1, x_2, \cdots, x_n) + \frac{1}{r}\sum_{i=1}^{m}\psi_i^2 + r^2\sum_{j=1}^{p}\frac{1}{\phi_j^S} + \frac{1}{r}\sum_{j=1}^{p} < \phi_j >^2 \quad .$$

Here the symbol < > has the previous meaning indicating an unsatisfied constraint, while superscript S indicates a satisfied one. The process begins with $r = 1$ and a sequence of optimization problems are solved in which $r$ is progressively reduced. To justify this expression, consider the effect on a two-dimensional problem with one inequality constraint. If the interior term is considered, $(r^2/\phi_j^S)$, with $r = 1$, it will push the surface upwards asymptotically to the constraint line, beginning the distortion some distance into the feasible region since, as the constraint line is approached, $\phi_j^S$ becomes smaller and smaller. It is apparent that a false constrained optimum point will be established well away from the true location at the constraint line. As $r$ becomes smaller, the effect of $r^2/\phi_j^S$ is increasingly reduced, the surface is sharpened, and the false constrained optimum point will approach the true one.

The external term $< \phi_j >^2 /r$ has a similar effect in the exterior region. With $r = 1$ , there is a gentle distortion of the surface remote from the constraint line which becomes small as $\phi_j$ approaches zero at the constraint line. A false unconstrained optimum point is created near to the feasible region. Subsequent reductions in $r$ increase the effect of the penalty term, sharpening the curve, and shifting the false, unconstrained optimum toward the true constrained location.

### D.3. Miscellaneous Methods

The group of miscellaneous methods in the **OPTIONS** package are drawn from a variety of sources and have no particular pattern or approach. However, they are sometimes remarkably effective on some problems and should not be overlooked. This part of the library also allows the user to add his or her own optimization codes.

#### D.3.1. NAg Methods

Early versions of the NAg maths library[15] contained three separate optimizers for dealing with unconstrained, variable bound constrained and generally constrained problems. Since mark 12 of the library these have been combined into an improved, general purpose routine called E04UCF[5]: this is the routine supported by the package, although links to the earlier routines may be provided if desired.

E04UCF uses a sequential quadratic programming (SQP) algorithm in which the search direction is the solution of a quadratic programming (QP) problem. The algorithm treats bounds, linear constraints and non-linear constraints separately.

#### D.3.2. Bit Climbing (BC)

A bit climbing algorithm is available[12]. This method works with a binary representation of the design variables and proceeds by flipping the bits of the design vector in random order, keeping changes if they improve the solution. It carries out cycles of flips with as many flips per cycle as there are bits in the problem. When an entire cycle of single bit flips makes no improvement it then starts flipping as many bits as there are variables in the problem until an improvement is made. It then reverts to single bit flips on the next cycle. Unlike simulated annealing it only retains improved solutions. The multi-bit flips are vital to prevent the optimizer becoming a local hill climber in multi-dimensional spaces and this is a major modification from the original reference. It stops when the number of function evaluations specified by the user is exhausted. This is a very simple optimizer which is controlled only by the binary precision of the search and the random number sequence in use.

Infeasible solutions are rejected by using either a simple one-pass method or by using the Fiacco and McCormick internal and external method. In the first method the objective function is set to be very high if any constraint is violated while in the second, infeasible constraints are penalised by squaring them and dividing by a penalty quantity before adding them to the objective function. The dividing penalty is given by raising the penalty index $r$ to the power of the cycle number number minus two (n.b., this is set to one on the first and second generations, $r$ on the third, $r^2$ on the fourth and so on). In addition *satisfied* inequality constraints are inverted and *multiplied* by the penalty quantity squared before also being added to the objective function. This combination starts to penalize the objective as it approaches the boundary of feasibility from the inside as well as hitting infeasible solutions. Moreover the penalty applied is slight to start with and increasingly more severe as the search continues. This method is not suitable for problems without bounds on *all* the variables, consequently it will not handle any of the NAg example problems. It is, however, capable of dealing with discrete design variables which can be useful.

### D.3.3. Dynamic Hill Climbing (DHC)

A dynamic hill climbing algorithm is available[9]. This method uses a series of gradient descent local searches to find individual optimal solutions, which are then used to find new starting points well away from previously found solutions, i.e., it aims to locate as many individual optimal solutions as possible with each one being found by a rapid local search. This works very well for multi-modal problems with a moderate number of peaks, relatively smooth surfaces and less than ten dimensions. It is not able to cope with problems where there are very many near optimal solutions to deal with. It should also be noted that this method always uses slightly more function evaluations than specified when starting the search as it is allowed to complete the current hill climb before finishing. This aspect of the methods should be allowed for when using the method.

This is an unconstrained search method and so the same penalty functions as described for the Bit Climbing routine are applied to drive the search away from infeasible solutions. It is not suitable for problems without bounds on *all* the variables, consequently it will not handle any of the NAg example problems. It is often compared to the GA to which it forms a credible alternative in some cases.

### D.3.4. Population Based Incremental Learning (PBIL)

A population based incremental learning algorithm is available[13]. This method is derived from work on the genetic algorithm but does not use cross-over between population members. Instead it maintains a probability vector which is a template that governs how new designs are produced. This template starts with probabilities of 0.5 for each bit in a binary encoding and it is then evolved towards probabilities of 1.0 or 0.0 as each bit is seen to need to be either on or off. It is governed by a learning rate parameter which controls the rate of change of the probability vector. This is updated once per generation depending on the best member seen in the generation and also when an elite (best-so-far) solution is seen. It also allows for random mutations of the vector once per generation.

This is an unconstrained search method and so the same penalty functions as described for the Bit Climbing routine are applied to drive the search away from infeasible solutions. It is not suitable for problems without bounds on *all* the variables, consequently it will not handle any of the NAg example problems. This routine is also capable of dealing with discrete design variables which can be important. It is often compared to the GA to which it forms a credible alternative in some cases.

This routine saves population statistics using the library routine DATAAN with a base name of 'PL' if suitable variables exist within the data-base. The style file "opt_pl" is suitable for plotting some of these statistics against population number if the population numbers are stored in the array WORK.

### D.3.5. Design of Experiments (DOE)

A design of experiments based optimizer is also available using pure random numbers, $LP_\tau$ pseudo random sequences (currently allows up to 25 variables and 16,000 different evaluations)[11], a central composite full factorial design augmented with $LP_\tau$ pseudo random sequences, a face centred cubic full factorial design augmented with $LP_\tau$ pseudo random sequences or Latin hypercubes. These methods are designed for use where only very few trials may be made relative to the size of the search space. In all cases a series of test points is decided at the outset and these points simply evaluated with the best result being returned as the solution. They are thus extremely robust and very easy to use with almost no parameters. If pure random numbers or Latin hypercubes are used the sequence can be readily varied if multiple runs are needed. With the $LP_\tau$ sequences a given sequence can be entered at any point but this is not recommended if only a very few points are being used. The aim of the $LP_\tau$ sequences is to give more uniform coverage of the search space than pure random numbers or Latin hypercubes do but otherwise they work in a similar fashion. The central composite design is the same as the face centred cubic design except that the corners of the cube are pulled inwards towards the centre of the design space in such a way that they are $1/\sqrt{2}$ of the full distance apart. The face centred cubic design simply places the calculations at the middle and extremes of the ranges of the variables (centroid of the hypercube first, followed by vertices and then centroids of the faces). Both of these full factorial designs are primarily designed for use with response surface curve fitting methods. The Latin hypercube process is very similar to pure random numbers except that the domain is initially split into a regular grid of hypercubes and then fresh random points are only accepted if they lie in previously unoccupied cubes. The process also ensures that at least 0.5% of the hypercubes reamin unused to ensure that the test design process is not overly time consuming. This routine is capable of dealing with discrete design variables. This is acheived by simply moving all the points in the test series to the nearest discrete values. If large numbers of trials are used this can lead to some duplication of trial points. If the appropriate operating system support is available this routine can run multiple function evaluations in parallel.

#### D.3.6. Numerical Recipes Methods (Num_Rcp)

Some of the routines discussed in the Numerical Recipes fortran cookbook[4] are available within the **OPTIONS** package. The interface provided allows use of that book's routines called DFPMIN (Broyden-Fletcher-Goldfarb-Shanno variant of Davidon-Fletcher-Powell minimization), FRPRMN (Fletcher-Reeves-Polak-Ribiere minimization) and POWELL (Powell method discarding the direction of largest decrease). Both the routines DFPMIN and FRPRMN require information on objective function gradients. When needed these are synthesized numerically by finite differencing with a fixed small step length. (N.b., by using the OPT_GRADNT feature of the package it is possible to use directly gradient calculations in place of these finite difference calculations.)

As described in the reference the routines used are for dealing with unconstrained problems. Here constraints are handled by the use of penalty functions as described for the Bit Climbing routine which are applied to drive the search away from infeasible solutions.

### D.4. OPTIVAR Methods

The OPTIVAR package contains various optimization methods in a format enabling simple use of a variety of techniques. Of these, the constrained and unconstrained nonlinear methods are relevant to the **OPTIONS** package. The constrained nonlinear methods available are:

(1) minimization by the method of successive linear approximation (APPROX) and

(2) minimization by random exploration with shrinkage (RANDOM).

The unconstrained nonlinear methods are a group of different strategies all using the same interchangeable penalty functions to deal with constraint violations:

(1) minimization by adaptive random search (ADRANS);

(2) minimization by Davidon, Fletcher, Powell strategy (DAVID);

(3) minimization by Fletcher's 1972 method (FLETCH);

(4) minimization by Jacobson and Oksman method (JO);

(5) minimization by Powell's direct search method (PDS);

(6) minimization by Hooke and Jeeves direct search (SEEK) and

(7) minimization by the simplex method (SIMPLX).

The penalty functions available are:

(1) One pass external function (OPTIM1);

(2) Fiacco-McCormick combined external and internal function[25] (OPTIM2);

(3) Powell's function[26] (OPTIM3) and

(4) Schuldt's function[27] (OPTIM5).

All of these methods are briefly described by Siddall[2], where further references to the original works may also be found. When using just the OPTIVAR package, the two constrained methods are usually adopted together with one of the unconstrained methods (often SEEK) combined with the first two penalty functions (OPTIM1 and OPTIM2). These represent four different strategies: APPROX is the quickest method if the function is smooth enough to be approximated to a linear one in the region of investigation. It tests the vertices of the $n$-dimensional form created from the objective function allowing for planes formed by linearizing the various constraints. RANDOM is a random "shotgun" search method, it is very slow, but also potentially the most reliable. If used in its purest form it will always find the optimum and so may be used as a reference for the evaluation of the other strategies (a shrinkage mechanism is invoked by default with this method to narrow the area for searching but this introduces the risk of finding a false optimum; see also the OPTIVAR routine ADRANS which can sometimes carry out more efficient random searches). Finally, one of the unconstrained nonlinear methods is usually selected. The Hooke and Jeeves method SEEK is usually chosen since it is both well known and typical of an heuristic approach. It is a direct search strategy that tests variables systematically and then decides the directions and sizes of steps to be taken using various comparisons. The penalty functions employed are basically of two kinds: a one pass external function, which penalizes the function only when the constraints are violated, and then extremely strongly; or multiple pass functions which penalize the function both inside the feasible region as the constraints are being approached as well as in the unfeasible area. The Fiacco-McCormick, Powell's and Schuldt's functions are multiple pass functions having slightly different formulations, but all rely on penalties that become increasingly severe as repeated optimizations are carried out. These later types of penalty function are useful when the function being optimized cannot cope with variables that lead to unfeasible results, making the system "crash". However, since they are mild in action and try to avoid going near unfeasible regions of the function where optima often lie, this approach tends to be slower than the first. For robust systems, i.e., systems that can cope with calculations in unfeasible areas, the one pass external function usually proves to be the quickest.

### D.4.1. Method of Successive Linear Approximation (APPROX)

The method employed is a successive linear approximation developed by Griffith and Stewart[28]. Starting at an initial point $(x_1^0, x_2^0, \cdots, x_n^0)$ the functions are approximated by expansion in a Taylor's series about $x_i^0$ in which terms above linear are dropped. As the problem has been converted to linear form, it can be solved by linear programming. The linearization is applied over a finite range that is subsequently moved and reduced as the optimization proceeds. The constraints, which are also linearized, are used to further reduce the linearized region. This method is very fast but has final convergence problems. These problems are exacerbated by singularities, excessive constraints or highly non-linear functions. It is also expensive on problems with many dimensions.

### D.4.2. Random Exploration with Shrinkage (RANDOM)

This method consists of a random search for the minimum, using a shotgun technique, with iterative shrinkage[29]. Random points for each variable $x_1$ to $x_n$ are generated from the expression $x_i = l_i + r_i(u_i - l_i)$ where $l_i$ is the estimated lower limit for $x_i$, $u_i$ is the estimated upper limit to $x_i$ and $r_i$ is a random number uniformly distributed between zero and one. Any generated point that violates an inequality constraint is discarded before the objective function is evaluated. If the constraints are violated a certain number of times consecutively, by default 300, the process stops.

The search is begun by evaluating a number of random points using the above equation, this number being a multiple of the number of variables. From these, the best results are selected and used as the basis for a new and shrunken range for each variable. The number of best results is defined by dividing the number of random points by a shrinkage factor. Within this new space, new random points are evaluated. These, plus the previous best results, are sorted to yield a new set of best results and a new shrunken space. The process is repeated until the range of each variable is acceptably small, or until the range has been shrunken a certain number of times, 1000 being the default. If any set of random points does not include one from near the optimal region the shrinkage mechanism can reject the area around the optimum and thus carries the risk of finding a false optimum. However, this mechanism does significantly improve the speed of the search, particularly over high dimensional spaces. When the problem is highly constrained shrinkage must be treated with caution.

### D.4.3. Adaptive Random Search (ADRANS)

This method uses an adaptive random search strategy followed by accelerated pattern moves. Since it does not hang up on boundaries it is a good method for checking the output of other techniques. It is quite a slow method but sometimes works when all others fail[30, 31]. It is also sometimes faster than the OPTIVAR routine RANDOM when carrying out random searches.

### D.4.4. Davidon, Fletcher, Powell Strategy (DAVID)

This method uses the Davidon, Fletcher, Powell gradient search in which at the step $k + 1$ the new value of an independent variable is given by

$$x_i^{k+1} = x_i^k + \lambda_i^k d_i^k$$

where $\lambda_i^k$ defines an optimum step length and $d_i^k$ is a function of the partial derivatives at $x_i^k$ and all of the derivatives at the previous steps. The partial derivatives are evaluated by numerical calculations[32].

### D.4.5. Fletcher's 1972 Method (FLETCH)

This method uses an algorithm developed by Fletcher and may call on either the Davidon, Fletcher, Powell or Broyden, Fletcher, Shannon methods. Partial derivatives are evaluated by numerical calculations[33].

### D.4.6. Jacobson and Oksman Method (JO);

This method is based on the assumption that an algorithm devised to work well for a homogeneous function will tend to be successful for almost any function[34].

### D.4.7. Powell's Direct Search Method (PDS)

This method is a direct search using the conjugate direction approach with quadratic convergence. The basic strategy is very simple: a search is made in each co-ordinate direction until a local minimum is found; after all co-ordinate directions have been searched, a move is made which finds a local minimum in the direction of the resultant of the previous co-ordinate searches. This is repeated except that in each iteration one co-ordinate direction is replaced by the last move direction. This routine is based on the routine VA04A produced by AERE Harwell[35].

### D.4.8. Hooke and Jeeves Direct Search (SEEK)

This method uses the following heuristic algorithm for minimization[36]:

(1) An arbitrary starting point must be selected, called the initial base point, at which the objective function $U$ is evaluated.

(2) An exploratory search is begun by increasing $x_1$ by a predetermined step length. If this improves the value of $U$, it is retained, if not, a corresponding reduction is made. If both fail, no change is made. A similar exploration is made for $x_2$ and so on. The final result of such an exploratory search is called a base point.

(3) A pattern move is made by changing each variable from the last base point an amount equal to the difference between the new and previous base point. This difference will commonly include a previous pattern move.

(4) If the pattern move fails to improve $U$, it is cancelled and replaced by a new search. If it succeeds, it is followed by a new search.

(5) The iteration continues until the exploratory search fails to locate a better point. The step length is then reduced an arbitrary amount, and the search is repeated. After each failure the step length is reduced an additional fraction until it reaches some predetermined minimum, when it is assumed that the optimum has been reached. When using the one pass external penalty function OPTIM1, the algorithm uses 100 random shotgun points when it fails to find a better point, to overcome stalling.

### D.4.9. Simplex Method (SIMPLX)

This method sets up a set of $n + 1$ points in $n$-dimensional space called the *simplex*. It gropes towards the optimum by flipping, expanding or contracting the simplex, the logic used being determined by the evaluation at each corner of the simplex. In this logic three parameters are required: an acceleration factor $\lambda(\lambda > 1)$, a contraction factor $\beta(0 < \beta < 1)$ and a step length factor $\alpha$, see Nelder and Meade[37].

### D.4.10. Remarks

These various approaches have a number of advantage and disadvantages. The random searches, RANDOM and ADRANS, as has already been mentioned, may be very slow, particularly if the feasible region is rather restricted, and there will then be a high risk of a large number of infeasible trials. On the other hand, they do not hang up on false or local optima. For this reason, they are good methods for checking the results of the other approaches within the OPTIVAR package. They have, however, been largely superseeded by the modern stochastic methods described in subsequent sections. Care must also be taken with RANDOM to ensure that the shrinkage mechanism does not reject the true optimum. With APPROX, if the function and boundaries formed by the constraints are not amenable to linear approximation, although finding the optimal region quickly, it will have great difficulty in converging. Testing of APPROX has shown that even functions of the sort found in typical design problems can sometimes be successfully approximated as linear over the small ranges required. However, highly constrained functions or those containing singularities are not handled well. With SEEK, the Hooke and Jeeves method, the main difficulty arises in highly constrained problems. The search can get stuck on constraints because of the fixed orientation of its search co-ordinates. If the contour lines and the inequality constraint lines happen to have certain orientations, and the search approaches from particular directions, the search may become stalled. A new starting point may well alleviate the problem. The multi-pass, "soft" penalty functions may also be helpful in extending the search. A local random search may help the search to jump out of trouble and this is included by default with the one pass penalty function. Both APPROX and SEEK rely on calculating the local slope of the objective function to indicate the direction of the optimal solution. If the slope is misleading they then tend to get stuck at local sub-optimal points.

## D.5. Schwefel Library Methods

The Schwefel library, as supplied on a floppy disk with his 1995 book[3], contains various optimization methods in a format enabling simple use of a variety of techniques. These are all constrained or unconstrained nonlinear methods and all are relevant to the **OPTIONS** package. The constrained nonlinear methods available are:

(1) minimization using Rosenbrock's rotating co-ordinate search[38] (ROSE);

(2) minimization by the complex strategy of M.J. Box[39] (COMP);

(3) minimization using Schwefel's two-membered evolution strategy (EVOL) and

(4) minimization using Schwefel's multi-membered evolution strategy (KORR).

The unconstrained nonlinear methods in the library, as implemented in the **OPTIONS** package, may be used with a choice of two interchangeable penalty functions to deal with constraint violations. The methods are:

(1) minimization by repeated application of a one-dimensional Fibonacci search (FIBO);

(2) minimization by repeated application of a one-dimensional Golden section search (GOLD);

(3) minimization by repeated application of a one-dimensional Lagrangian interpolation search (LAGR);

(4) minimization by Hooke and Jeeves direct search (HOJE);

(5) minimization using the strategy of Davies, Swann and Campey with Gram-Schmidt orthogonalization (DSCG);

(6) minimization using the strategy of Davies, Swann and Campey with Palmer orthogonalization (DSCP);

(7) minimization by Powell's strategy of conjugate directions (POWE);

(8) minimization by the Davidon, Fletcher, Powell strategy (DFPS) and

(9) minimization by the simplex strategy of Nelder and Meade (SIMP)[37].

The penalty functions available are a simple one-pass external function (OPTIM1) and the Fiacco-McCormick combined external and internal function[25] which is applied over a series of sequential unconstrained minimizations (OPTIM2).

The various optimization methods are fully described by Schwefel[3], where further references to the original works may also be found. Notice that several of the classical methods provided by Schwefel are common to both this library and the OPTIVAR package, although details of their exact implementations differ. Those methods that are common to both suites do not, therefore, always deliver the same results. Also, the evolution strategies provided by Schwefel should be considered the definitive sources for these methods, since the method was originally developed by Schwefel and his co-workers. It has, however, been observed that they are sometimes outperformed by the evolution strategy developed specifically for the **OPTIONS** package (ES).

#### D.5.1. Rosenbrock's Rotating Co-ordinate Search (ROSE)

This method[38] works in a similar manner to the Hooke and Jeeves search but additionally allows the directions of search to alter so that it is not restricted to the co-ordinate system based on the individual variable directions.

#### D.5.2. Complex Strategy of M.J. Box (COMP)

This method[39] uses a constrained version of the simplex algorithm[37] which is termed a "complex". It uses more vertices and expansion of the polyhedron at each normal reflection. It is not considered efficient when more than five variables must be dealt with.

#### D.5.3. Two-membered Evolution Strategy (EVOL)

The two-membered evolution strategy is the result of the early work of Schwefel and his collaborators. It has largely been superseeded by the multi-membered strategy, although it has rather fewer parameters and thus tends to require less tuning.

#### D.5.4. Multi-membered Evolution Strategy (KORR)

The multi-membered evolution strategy represents Schwefel's major contribution to the subject of evolutionary optimization. It is discussed at some length in his book[3] and also in the subsequent section of this appendix on modern stochastic optimizers.

#### D.5.5. Repeated Fibonacci Search (FIBO)

This method is a based on the repeated application in successive search directions of a simple one-dimensional Fibonacci search.

#### D.5.6. Repeated Golden Section Search (GOLD)

This method is a based on the repeated application in successive search directions of a simple one-dimensional Golden Section search.

#### D.5.7. Repeated Lagrangian Interpolation (LAGR)

This method is a based on the repeated application in successive search directions of a simple one-dimensional Lagrangian interpolation.

#### D.5.8. Hooke and Jeeves Direct Search (HOJE)

This method uses the same heuristic algorithm[36] as the OPTIVAR routine SEEK although its implementation is slightly different.

#### D.5.9. Davies, Swann and Campey Search Method with Gram-Schmidt Orthogonalization (DSCP)

In this method[40] small changes in all variables are tested one at a time to establish a suitable direction for a simple constant direction search, followed by an orthogonalization to establish a new set of rotated co-ordinates in which to carry out the small step tests. This version uses Gram-Schmidt orthogonalization.

#### D.5.10. Davies, Swann and Campey Search Method with Palmer Orthogonalization (DSCP)

In this method[40] small changes in all variables are tested one at a time to establish a suitable direction for a simple constant direction search, followed by an orthogonalization to establish a new set of rotated co-ordinates in which to carry out the small step tests. This version uses Palmer orthogonalization.

### D.5.11. Powell's Direct Search (POWE)

This method[35] uses the same conjugate direction approach as the OPTIVAR routine PDS although its implementation is slightly different.

### D.5.12. Davidon, Fletcher, Powell Strategy (DFPS)

This method[32] uses the same Davidon, Fletcher, Powell gradient search as the OPTIVAR routine DAVID although its implementation is slightly different.

### D.5.13. Simplex Method (SIMP)

This method uses the same Nelder and Meade algorithm[37] as the OPTIVAR routine SIMPLX although its implementation is slightly different. To avoid a name clash with the OPTIVAR package the routine SIMP in the Schwefel library *must* be renamed as SSIMP.

### D.5.14. Remarks

In addition to those points made at the end of the section on the OPTIVAR library concerning the methods common to both the OPTIVAR and Schwefel libraries, the following further remarks should be born in mind when using the routines provided by Schwefel:

(1) the Schwefel library of methods does not appear to have been as extensively tested as the OPTIVAR library and its routines tend to stall or fail more frequently (as supplied it has a number of faults which can be fairly easily corrected before being used†);

(2) the Rosenbrock rotating co-ordinate search (ROSE) tends to stall very easily and must also be started from a feasible point, it does, however, sometimes perform spectacularly well, even on high dimension problems[38];

(3) the convergence of the repeated Lagrangian interpolations (LAGR) is often very poor and is almost always outperformed by the Fibonacci (FIBO) and Golden Section searches (GOLD);

(4) as implemented by Schwefel, using a modified convergence criterion, the Davidon, Fletcher, Powell strategy (DFPS) is easily stalled and is also rather susceptible to changes in the estimated solution value used in the convergence tests;

(5) minimization using the simplex strategy of Nelder and Meade (SIMP)[37] is badly behaved when trying to carry out final converge to solutions, especially those defined by constraints - therefore large numbers of trials should not be allocated when first using this method and also convergence errors should be anticipated;

(6) minimization using Schwefel's two-membered evolution strategy on highly multi-modal functions is improved by increasing the step size refinement ratio, given as a sub-command to the package, from 0.85 to 0.99 or even 0.999.

## D.6. Modern Stochastic Methods

All these methods generate sequences of random points but use some overall control strategy to link the acceptance of new points to the progress of the search. Since they produce new points in this way they do not use information about the local slope of the objective function and are thus not prone to stalling in false optima. They are all essentially unconstrained methods and so make use of penalty functions to deal with constraints. Currently, only the one pass external (OPTIM1) and Fiacco and McCormick[25] (OPTIM2) functions are implemented for these methods. When using the one pass function this means that infeasible points are simply heavily penalized. The multi-pass functions are handled by slowly modifying the penalty as the optimization proceeds, so that near the beginning of the search the functions are only slightly

---

† - the format statements in the routine KORR as supplied on the disk are in German, the English version are given in the listing in Schwefel's book; line 112 in the routine POWE should read "112  IF(FP-F)37,37,91"; line 35 in the routine POWE can give rise to negative square roots and should be replaced by "35  IF(FP-F.LT.0.0D0)GOTO 78" followed immediately by " DDMAG=0.4*DSQRT(FP-F)"; the routine GRAD can generate invalid gradients by using zero length divisors, an improved version is available with the **OPTIONS** package and can be invoked by commenting out this routine in the Schwefel supplied library; finally, to avoid routine name clashes with the OPTIVAR package the routine SIMP should be renamed SSIMP, if this is not done errors will occur when trying to invoke this routine from the Schwefel "Schwefel_Optimizers" sub-menu.

penalized while at the end the penalties are most severe. All the methods run for a fixed number of trials as they do not incorporate convergence tests. As encoded here, they all discretize the search space in binary arithmetic and this enables them to readily deal with discrete valued design variables. By default they use 12 bit arithmetic for each variable (although this may be varied between eight and 16 bits).

### D.6.1. Genetic Algorithm (GA)

The G.A. used is fairly typical of those discussed by Goldberg[6] except that it incorporates a version of MacQueen's Adaptive KMEAN algorithm[41] which has been applied recently with some success to multi-peak problems[42]. It is advisable to disable this clustering/niche forming algorithm if it is known that the objective function does not have multiple peaks or troughs. In the form used here the G.A. has 13 control parameters that the user can select before applying the method (defaults are available). It works with up to 16 bit binary encoding (although parameters that are selected from a number of fixed possibilities use only the minimum necessary number of bits), an elitist survival strategy which ensures that the best of each generation always enters the next generation, has seven major control parameters, two constraint penalty functions and optional niche forming. The penalty functions are either the one pass external function with a large, fixed penalty mentioned above (the default) or a Fiacco and McCormick function with increasing severity on each generation.

In this case the Fiacco and McCormick function is set up to fit naturally within the G.A. It modifies the objective function as follows:

$$U_p = U(x_1, x_2, \cdots, x_n) + r^{2.\max(1, n/N_{\text{pop}}-2)} \sum_{j=1}^{p} \frac{1}{\phi_j^S} + \frac{1}{r^{\max(1, n/N_{\text{pop}}-2)}} \sum_{j=1}^{p} < \phi_j >^2 \quad .$$

Again, $< \phi_i >$ are the violated constraint conditions and $\phi_i^S$ those that are satisfied. $r$ is the penalty quantity, *less than unity*, chosen to suit the problem in hand, $n$ the current evaluation number and $N_{\text{pop}}$ the number of members in each generation. Thus, to begin with, when $n/N_{\text{pop}}$ is small, violated constraints are only slightly penalized along with satisfied ones that are near the constraint boundaries, effectively warning the optimizer of the presence of boundaries while allowing their exploration. As the optimization proceeds and the number of evaluations increases, $n/N_{\text{pop}}$ grows causing $r^{\max(1, n/N_{\text{pop}}-2)}$ to become exponentially small, severely penalizing the violated constraints and removing the effect of the term involving the satisfied ones, causing the optimizer to chose feasible solutions where $U_p$ reverts to $U$. Notice that here the division $n/N_{\text{pop}}$ is carried out in integer arithmetic causing this quantity to be constant over members of the same generation. By also subtracting 2, but preventing the result from being less than unity, the penalty is deliberately held low during the first three generations of the method, giving the G.A. chance to get started before the penalizing function takes effect The use of an evolving constraint penalty fits particularly naturally within the general scheme of the G.A. yet is not widely used, perhaps because most applications of the G.A. to date have been to unconstrained problems.

The main parameters used in the G.A. here are:

- $N_{\text{gen}}$ — the number of generations allowed (default 10);
- $N_{\text{pop}}$ — the population size or number of trials used per generation which is therefore inversely related to the number of generations given a fixed number of trials in total (default 100);
- P[best] — the proportion of the population that survive to the next generation (default 0.8);
- P[cross] — the proportion of the surviving population that are allowed to breed (default 0.8);
- P[invert] — the proportion of the surviving population that have their genetic material re-ordered (default 0.5);
- P[mutation] — the proportion of the new generation's genetic material that is randomly changed (default 0.005);
- a Proportionality Flag — which selects whether the new generation is biased in favour of the most successful members of the previous generation or alternatively if all P[best] survivors are propagated equally (default TRUE) and
- the Penalty Function Choice — together with the value of $r$ in the case of the Fiacco and McCormick method (0.5 is a sensible value if no other information is available, but care should be taken to ensure sufficient

generations are in use for the penalty to become suitably severe by the end of the search - the method prints out the final penalty applied to enable this aspect of the process to be checked).

The version of the adaptive KMEAN algorithm used here is controlled by:

$D_{\min}$ the minimum non-dimensional Euclidean distance between cluster centres, with clusters closer than this being collapsed (default 0.1);

$D_{\max}$ the maximum non-dimensional Euclidean radius of a cluster, beyond which clusters sub-divide (default 0.2);

$N_{\text{clust}}$ the initial number of clusters into which a generation is divided (default 25);

$N_{\text{breed}}$ the minimum number of members in a cluster before exclusive inbreeding within the cluster takes place (default 5) and

$\alpha$

$\alpha$ the penalising index for cluster members which determines how severely members sharing an over-crowded niche will suffer, with small numbers giving less penalty, i.e., the objective functions of members of a cluster of $m$ solutions are scaled by $m^{\min(\alpha,1)}[1-(E/D_{\max})^{\alpha}] + (E/D_{\max})^{\alpha}$, where $E$ is the Euclidean distance of the member from its cluster centre, which is always less than $D_{\max}$, moreover when $E = D_{\max}$ or $\alpha = 0$ no penalty is applied, (default 0.5, setting this parameter negative disables the clustering and this should be done if it is known that the function being dealt with is not multi-modal),

The routine is rather slow and inaccurate for problems with few variables but comes into its own as the number of variables becomes higher than five. It is also not suitable for problems without bounds on *all* the variables, consequently it will not handle any of the NAg example problems. Currently the GA seems to the best of the stochastic methods.

The routine saves population statistics using the library routine DATAAN with a base name of "GA" if suitable variables exist within the data-base. The style file "opt_ga" is suitable for plotting some of these statistics against population number if the population numbers are stored in the array WORK. If the appropriate operating system support is available this routine can run multiple function evaluations in parallel.

### D.6.2. Simulated Annealing (SA)

This algorithm[7] works by using random mutations which are retained if they improve the solution and sometimes if they do not depending on a Boltzmann probability function. This function is

$$\exp\left(-\frac{\mathrm{d}U}{T\,\mathrm{d}U_{FIRST}}\right) > R$$

where $R$ is a uniformly probable random number in the interval 0-1, $\mathrm{d}U$ is the change in objective function value, $\mathrm{d}U_{FIRST}$ is the absolute value of the first, unpenalized, *change* in the objective function (i.e., from the starting point to the next position tried) and $T$ is the non-dimensional annealing temperature which is $\gg 1$ at high temperature and tends to zero at low temperature. $T$ is progressively lowered in logarithmic fashion as the solution moves on, i.e., various solutions are tried at each effective temperature. The number of temperatures and the number of trials at each temperature control the total number of calculations carried out. These are set so that the number of temperatures, $N_T$, is equal to the number of iterations raised to some power, $P_T$, and the number of trials at each temperature is the total number divided by the number of temperatures, i.e., $N_T = \text{number\_trials}^{P_T}$. $P_T$ must thus lie between zero and one, with zero implying a simple random search at one temperature and one a single trial at each temperature. The temperature schedule is controlled by two parameters, $P_W$ and $P_C$ such that

$$T = P_W^{\left(\frac{N_T}{P_C} - K\right)}, \qquad K = 1, N_T \quad .$$

Here $P_W$ is set to lie in the range 1-10 and $P_C$ 0.1-10, thus large values of $P_W$ give a wide range of temperatures, while large values of $P_C$ bias this range to low values (a $P_W$ of one fixes all temperatures to be the same while $P_C$ of two biases the temperatures equally between hot and cold). If definite start and finish temperatures, $T_{start}$ and $T_{finish}$, are desired then set

$$P_W = \left( \frac{T_{start}}{T_{finish}} \right)^{\frac{1}{N_T - 1}} \qquad \text{and} \qquad P_C = \frac{N_T \log(P_W)}{\log(T_{start} P_W)} \quad .$$

This is an unconstrained search method and so the same penalty functions as described for the Genetic Algorithm routine are applied to drive the search away from infeasible solutions. When using the modified Fiacco and McCormick function the division into separate annealing temperatures takes the place of the different generations of the Genetic Algorithm. This version also has an elitist strategy which remembers the best solution ever seen and ensures this is delivered as the answer although it is not used to control the annealing process.

This routine is rather slow and inaccurate for problems with few variables but comes into its own as the number of variables becomes higher than five. It is also not suitable for problems without bounds on *all* the variables, consequently it will not handle any of the NAg example problems. This routine is also capable of dealing with discrete design variables which can be important. It is often compared to the GA to which it forms a credible alternative. It is very susceptible to the annealing schedule used and the default one is merely a reasonably robust general purpose version. A specialized schedule can usually be derived for any given problem which improves the speed of the method. Currently, SA without a problem specific cooling schedule seems to be the worst of the stochastic routines.

### D.6.3. Evolutionary Programming (EP)

An evolutionary programming algorithm is available[8] where evolution is carried out by forming a mutated child from each parent in the population with mutation related to the objective function so that successful ideas are mutated less. The objective functions of the children are then formed and a stochastic ranking process used to select the next parents from the combined set of parents and children. The best of the solutions is kept unchanged at each pass to ensure that only improvements are allowed. The mutation is controlled by a parameter which sets the order of the variation in the amount of mutation with ranking. A value of one thus gives a linear change, two a quadratic and so on (only positive values being allowed). In all cases the best parent is not mutated and almost all bits in the worst are changed.

The stochastic process for deciding which elements survive involves jousting each member against a tournament of other members selected at random (including possibly itself) with a score being allocated for the number of members in the tournament worse than the case being examined. Having scored all members of both parent and child generations the best half are kept to form the next set of parents. The average number of solutions in the tournament is set by a second parameter. An elitist trap ensures that the best of all always survives. This is an unconstrained search method and so the same penalty functions as described for the Genetic Algorithm routine are applied to drive the search away from infeasible solutions.

This routine is rather slow and inaccurate for problems with few variables but comes into its own as the number of variables becomes higher than five. It is also not suitable for problems without bounds on *all* the variables, consequently it will not handle any of the NAg example problems. This routine is also capable of dealing with discrete design variables which can be important. It is often compared to the GA to which it forms a credible alternative. It is susceptible to the tournament size and using half the population, which is the default, is a good starting point. The mutation index can be used to give greater or less exploration depending on if it is decreased below two or increased above it, respectively. Currently EP seems to lie in the middle of GA, SA and EP.

This routine saves population statistics using the library routine DATAAN with a base name of 'EP' if suitable variables exist within the data-base. The style file "opt_ep" is suitable for plotting some of these statistics against population number if the population numbers are stored in the array WORK.

### D.6.4. Evolution Strategy (ES)

An evolution strategy algorithm is available[10]. This method uses evolving populations of solutions in much the same way as the GA. However, it uses real value encoding and a sophisticated evolving vector of mutation parameters. The routine can be used with the mutated children replacing their parents, a so called $(\mu, \lambda)$-ES, or with the next generation being selected from the combined parents and children, a $(\mu + \lambda)$-ES. Cross-over can be discrete or intermediate on both the design vector and the mutation control vector. The version provided as ES does *not* use correlated mutations.

The algorithm works by forming a given number of random guesses and then attempts to improve on them, maintaining a number of best guesses as the process continues. This number is the population. Evolution is carried out by forming a child population by mating pairs of parents, followed by mutation. The next generation is then made up from good members of the old and child populations. Two methods can be used: either the best from the combined and child populations are kept or, alternatively, only the best from the child population are used, with any shortfall being made up from the best of the parents. In the latter case, if the best parent is better than the best child, it is preserved unchanged (elitism).

The breeding of new children can be discrete and then individual design vector values are taken from either parent randomly, or intermediate and the values are then the average of those of the parents. Separate controls for the breeding of the design vector and the mutation standard deviations are available so that these can use different approaches. The best of the solutions is remembered at each pass to ensure that the final result is the best solution of all.

The mutations are controlled by a vector of standard deviations which also evolves with the various population members. Each element in the design vector is changed by the addition of a random number with this standard deviation scaled to the range of the variable. The rate at which the S.D.'s change is controlled by a parameter of the method, $\Delta\sigma$ such that $\sigma_{i\mathrm{NEXT}} = \sigma_{i\mathrm{PREV}} e^{\mathrm{N}(0,\Delta\sigma)}$ where the $\sigma_i$ are the individual standard deviations and $\mathrm{N}(0, \Delta\sigma)$ is a random number with zero mean and $\Delta\sigma$ standard deviation. The $\sigma_i$ are additionally restricted to be less than 0.1 so that the changes to the design vector (in a non-dimensional unit hyper-cube) do not simply push the vector to its extreme values. The initial values of the $\sigma_i$'s are set to be 0.025.

This is an unconstrained search method and so the same penalty functions as described for the Genetic Algorithm routine are applied to drive the search away from infeasible solutions. It is not suitable for problems without bounds on *all* the variables, consequently it will not handle any of the NAg example problems. This routine is *not* capable of dealing with discrete design variables. It is often compared to the GA to which it forms a credible alternative in many cases.

This routine saves population statistics using the library routine DATAAN with a base name of 'ES' if suitable variables exist within the data-base. The style file "opt_es" is suitable for plotting some of these statistics against population number if the population numbers are stored in the array WORK.

# APPENDIX E

## Listing for Subroutines OPTFUN and OPTCON, version I

```
      SUBROUTINE OPTFUN(ERROR)
C
C-----------------------------------------------------------------------
C
C        Routine specific type declaration block
C
      LOGICAL ERROR
      DOUBLE PRECISION GETVVD,BRDTH,HEIGHT,AREA
C
C-----------------------------------------------------------------------
C
C        The main areas of routine code now follow.
C
C        n.b. breadth in mm and height in cm
C
      ERROR=.FALSE.
      BRDTH=MAX(1.0D-3,GETVVD('BREADTH',1,1,1,ERROR))
      IF(ERROR)GOTO 999
      HEIGHT=MAX(1.0D-3,10.0D0*GETVVD('HEIGHT',1,1,1,ERROR))
      IF(ERROR)GOTO 999
C
C        cross-sectional area
C
      AREA=BRDTH*HEIGHT
C
C        save results
C
      CALL SETVVD('AREA',1,1,1,AREA,ERROR)
      IF(ERROR)GOTO 999
C
C-----------------------------------------------------------------------
C
C        The remaining lines are standard routine exit code.
C
  999 RETURN
      END


      SUBROUTINE OPTCON(ERROR)
C
C-----------------------------------------------------------------------
C
C        Routine specific type declaration block
C
      LOGICAL ERROR,DEFLT
      DOUBLE PRECISION GETVVD,BRDTH,HEIGHT,FORCE,LENGTH,EE,GG,NU,FACTOR
     1 ,SIGMAB,TAU,DEFLN,FCRIT,HONB,SIGMAY
      REAL GETDEF
```

```
C
C-------------------------------------------------------------------------
C
C       The main areas of routine code now follow.
C
C       n.b. length & breadth in mm height in cm
C
      ERROR=.FALSE.
      BRDTH=MAX(1.0D-3,GETVVD('BREADTH',1,1,1,ERROR))
      IF(ERROR)GOTO 999
      HEIGHT=MAX(1.0D-3,10.0D0*GETVVD('HEIGHT',1,1,1,ERROR))
      IF(ERROR)GOTO 999
C
C       set parameters
C
      LENGTH=MAX(1.0E-3,GETDEF('LENGTH',1,1,1,1500.0,DEFLT))
      FORCE=GETDEF('FORCE',1,1,1,5000.0,DEFLT)
      FACTOR=GETDEF('FACTOR',1,1,1,2.0,DEFLT)
C
C       decide material type and set material properties
C
      EE=GETDEF('EE',1,1,1,216620.0,DEFLT)
      GG=GETDEF('GG',1,1,1,86650.0,DEFLT)
      NU=GETDEF('NU',1,1,1,0.27,DEFLT)
      SIGMAY=GETDEF('SIGMAY',1,1,1,200.0,DEFLT)
C
C       set material limits (tau-y = sigma-y / 2)
C
      CALL SETVVD('U_SIGMA-B',1,1,1,SIGMAY,ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('U_TAU',1,1,1,SIGMAY/2.0D0,ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('L_F-CRIT',1,1,1,FORCE*FACTOR,ERROR)
      IF(ERROR)GOTO 999
C
C       bending stress
C
      SIGMAB=6.0*FORCE*LENGTH/(BRDTH*HEIGHT*HEIGHT)
C
C       shear stress
C
      TAU=1.5*FORCE/(BRDTH*HEIGHT)
C
C       tip deflection
C
      DEFLN=4.0*FORCE*LENGTH**3/(EE*BRDTH*HEIGHT**3)
C
C       twist buckling instability load
C
      FCRIT=(4.0*SQRT(GG*(BRDTH**3*HEIGHT+HEIGHT**3
     1 *BRDTH)*EE*(BRDTH**3*HEIGHT)
     2 /(144.0*(1.0-NU*NU)))/(LENGTH*LENGTH))
C
C      section aspect ratio
C
      HONB=HEIGHT/BRDTH
C
```

```
C         save results, limiting them to sensible numbers
C
      CALL SETVVD('SIGMA-B',1,1,1,MIN(1.0D15,SIGMAB),ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('TAU',1,1,1,MIN(1.0D15,TAU),ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('DEFLN',1,1,1,MIN(1.0D15,DEFLN),ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('F-CRIT',1,1,1,MIN(1.0D15,FCRIT),ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('H-ON-B',1,1,1,MIN(1.0D15,HONB),ERROR)
      IF(ERROR)GOTO 999
C
C-----------------------------------------------------------------------
C
C         The remaining lines are standard routine exit code.
C
  999 RETURN
      END
```

# APPENDIX F

## Terminal Interaction for Manual Example

User input follows on the lines starting with a "?". All other characters are output by the program except those in italics which are comments on the user input.

```
GENERAL PURPOSE OPTIMIZER PROGRAM


This program allows computerized optimization to be
carried out. To use the package you must provide various
subroutines and link them to this package using the "f77"
compiler. Having linked the program you can then use
the various commands available to set-up and solve your
specific problem.

If you need help try using the "help" command.



GENDAT Version A2.1 - Dated 30 Jul 1993


Data read from file opdata.d
Select optimizer option
```

? Full_Control , On — *turn on all GENDAT facilities*

```
Option :- Full_Control , On
Select optimizer option
```

? Set_Data_Types , * , Kill — *remove all existing structure*

```
Option :- Set_Data_Types , * , Kill
Sub-option :- Kill

Do you really wish to delete the ENTIRE data structure
```

? yes — *confirm deletion*

```
*** WARNING *** - data structure will be destroyed.
```

```
Are you sure

? yes                                                   confirm deletion
Select optimizer option

? Input_Data_Value , Data_Usage , Full                  add comments
Option :- Input_Data_Value , Data_Usage , Full
Input up to 10 sixty character lines describing the usage of this
data structure. Terminate the lines with a lone "!".
? OPTIMIZER data structure - 10/12/93
? This data set is used by the program "beam1" to
? carry out optimization calculations on a beam.
? First version with two variables and minimum
? data set.
?
?
?
?
?
Select optimizer option

? Set_Data_Types , Constraint , Add                     add new data type
Option :- Set_Data_Types , Constraint , Add
Sub-option :- Add

Select optimizer option

? Set_Data_Types , Parameter , Add                      add new data type
Option :- Set_Data_Types , Parameter , Add
Sub-option :- Add

Select optimizer option

? Set_Data_Types , Opt_Var , Add                        add new data type
Option :- Set_Data_Types , Opt_Var , Add
Sub-option :- Add

Select optimizer option

? Set_Data_Types , Obj_Funct , Add                      add new data type
Option :- Set_Data_Types , Obj_Funct , Add
Sub-option :- Add

Select optimizer option

? Set_Data_Types , Dependent , Add                      add new data type
Option :- Set_Data_Types , Dependent , Add
Sub-option :- Add

Select optimizer option

? Set_Data_Types , Limit , Add                          add new data type
Option :- Set_Data_Types , Limit , Add
Sub-option :- Add

Select optimizer option
```

```
? Add_Data_Record                                        add new variable
Option :- Add_Data_Record
Sub-option :- Name
Defining variable *
Input variable name, with no spaces, "*"s or "/"s.

? AREA                                                   give the variable's name
Sub-option :- Dimensions
Defining variable AREA
Input X, Y and Z array dimensions.

? 1, 1, 1                          give the variable's array subscript sizes
Sub-option :- Type
Defining variable AREA
Input variable type

? Obj_Funct                                              give the variable's type
Sub-option :- Units
Defining variable AREA
Input variable units

? mm^2                                                   give the variable's units
Sub-option :- Meaning
Defining variable AREA
Input variable meaning

? Cross-sectional area of the beam.                      give the variable's meaning
Sub-option :- Rec_No
Defining variable AREA
Select optimizer option

? Add_Data_Record
Option :- Add_Data_Record
Sub-option :- Name
Defining variable *
Input variable name, with no spaces, "*"s or "/"s.

? BREADTH
Sub-option :- Dimensions
Defining variable BREADTH
Input X, Y and Z array dimensions.

? 1, 1, 1
Sub-option :- Type
Defining variable BREADTH
Input variable type

? Opt_Var
Sub-option :- Units
Defining variable BREADTH
Input variable units

? mm
Sub-option :- Meaning
Defining variable BREADTH
Input variable meaning
```

```
? Breadth of the beam.
Sub-option :- Rec_No
Defining variable BREADTH
Input record number between 0 and 2

? 0
Record number set to     2
Select optimizer option

? Add_Data_Record
Option :- Add_Data_Record
Sub-option :- Name
Defining variable *
Input variable name, with no spaces, "*"s or "/"s.

? HEIGHT
Sub-option :- Dimensions
Defining variable HEIGHT
Input X, Y and Z array dimensions.

? 1, 1, 1
Sub-option :- Type
Defining variable HEIGHT
Input variable type

? Opt_Var
Sub-option :- Units
Defining variable HEIGHT
Input variable units

? cm
Sub-option :- Meaning
Defining variable HEIGHT
Input variable meaning

? Height of the beam.
Sub-option :- Rec_No
Defining variable HEIGHT
Input record number between 0 and 3

? 0
Record number set to     3
Select optimizer option
.
.
.
Select optimizer option

? Add_Data_Record
Option :- Add_Data_Record
Sub-option :- Name
Defining variable *
Input variable name, with no spaces, "*"s or "/"s.

? L_F-CRIT
Sub-option :- Dimensions
Defining variable L_F-CRIT
```

```
Input X, Y and Z array dimensions.

? 1, 1, 1
Sub-option :- Type
Defining variable L_F-CRIT
Input variable type

? Limit
Sub-option :- Units
Defining variable L_F-CRIT
Input variable units

? N
Sub-option :- Meaning
Defining variable L_F-CRIT
Input variable meaning

? Min. permissible critical force for twist buckling.
Sub-option :- Rec_No
Defining variable L_F-CRIT
Input record number between 0 and 24

? 0
Record number set to   24
Select optimizer option

? Add_Data_Record
Option :- Add_Data_Record
Sub-option :- Name
Defining variable *
Input variable name, with no spaces, "*"s or "/"s.

? WORK                                                     work space variable
Sub-option :- Dimensions
Defining variable WORK
Input X, Y and Z array dimensions.

? 1, 1, 1
Sub-option :- Type
Defining variable WORK
Input variable type

? Parameter
Sub-option :- Units
Defining variable WORK
Input variable units

?
Sub-option :- Meaning
Defining variable WORK
Input variable meaning

? Work space array.
Sub-option :- Rec_No
Defining variable WORK
Input record number between 0 and 25
```

```
? 0
Record number set to   25
Select optimizer option

? Input_Data_Value , BREADTH , Full                              insert numeric data
Option :- Input_Data_Value , BREADTH , Full
Input value of BREADTH Breadth of the beam. mm

? 0.30000000E+02                                                 give value
Select optimizer option

? Input_Data_Value , HEIGHT , Full
Option :- Input_Data_Value , HEIGHT , Full
Input value of HEIGHT Height of the beam. cm

? 0.70000000E+01
Select optimizer option

? Input_Data_Value , LENGTH , Full
Option :- Input_Data_Value , LENGTH , Full
Input value of LENGTH Length of the beam. mm

? 0.15000000E+04
Select optimizer option

? Input_Data_Value , FORCE , Full
Option :- Input_Data_Value , FORCE , Full
Input value of FORCE Force on the tip of the beam. N

? 0.50000000E+04
Select optimizer option

? Input_Data_Value , EE , Full
Option :- Input_Data_Value , EE , Full
Input value of EE Young's modulus. MPa

? 0.21662000E+06
Select optimizer option

? Input_Data_Value , GG , Full
Option :- Input_Data_Value , GG , Full
Input value of GG Modulus of rigidity. MPa

? 0.86650000E+05
Select optimizer option

? Input_Data_Value , NU , Full
Option :- Input_Data_Value , NU , Full
Input value of NU Poisson's ratio.

? 0.27000000E+00
Select optimizer option

? Input_Data_Value , SIGMA-Y , Full
Option :- Input_Data_Value , SIGMA-Y , Full
Input value of SIGMA-Y Yield stress. MPa
```

```
? 0.20000000E+03
Select optimizer option

? Input_Data_Value , FACTOR , Full
Option :- Input_Data_Value , FACTOR , Full
Input value of FACTOR Safety factor on the critical force for twist
buckling.

? 0.20000000E+01
Select optimizer option

? Input_Data_Value , L_BREADTH , Full
Option :- Input_Data_Value , L_BREADTH , Full
Input value of L_BREADTH Min. permissible beam breadth. mm

? 0.50000000E+01
Select optimizer option

? Input_Data_Value , U_BREADTH , Full
Option :- Input_Data_Value , U_BREADTH , Full
Input value of U_BREADTH Max. permissible beam breadth. mm

? 0.50000000E+02
Select optimizer option

? Input_Data_Value , L_HEIGHT , Full
Option :- Input_Data_Value , L_HEIGHT , Full
Input value of L_HEIGHT Min. permissible beam height. cm

? 0.20000000E+01
Select optimizer option

? Input_Data_Value , U_HEIGHT , Full
Option :- Input_Data_Value , U_HEIGHT , Full
Input value of U_HEIGHT Max. permissible beam height. cm

? 0.25000000E+02
Select optimizer option

? Input_Data_Value , U_DEFLN , Full
Option :- Input_Data_Value , U_DEFLN , Full
Input value of U_DEFLN Max. permissible tip deflection. mm

? 0.50000000E+01
Select optimizer option

? Input_Data_Value , U_H-ON-B , Full
Option :- Input_Data_Value , U_H-ON-B , Full
Input value of U_H-ON-B Max. permissible height to breadth ratio.

? 0.10000000E+02
Select optimizer option

? Store_Data , beam1.d
```
*define data name and save structure*
```
Option :- Store_Data , beam1.d
Data write to file beam1.d
Select optimizer option
```

```
? Print_Data                                          produce formatted print file
Option :- Print_Data , * , *
Data output will be sent to a new file for subsequent printing.
Input valid file name.

? beam1.txt                                           give print file name
Select optimizer option

? Show_Optimization                                   show results before first pass
Option :- Show_Optimization

*** WARNING *** - Variable U_SIGMA-B
This upper bound has no value set and has been ignored.
*** WARNING *** - Variable SIGMA-B
This variable has no constraints set and has been ignored.
*** WARNING *** - Variable U_TAU
This upper bound has no value set and has been ignored.
*** WARNING *** - Variable TAU
This variable has no constraints set and has been ignored.


=============================================================================

Status after      0 evaluations is :-

                    Trial Vector

   Lwr Bound        Vector        Uppr Bound    Variable (units)

  5.000000    <  30.00000    <  50.00000      BREADTH (mm)
  2.000000    <  7.000000    <  25.00000      HEIGHT (cm)

No of V. Boundary Violations =   0

Objective Function  NOT CALCULATED            AREA (mm^2)

                Constraints Vector

   Lwr Bound        Vector        Uppr Bound    Variable (units)

          Constraint NOT CALCULATED            DEFLN (mm)
          Constraint NOT CALCULATED            H-ON-B ()
          Constraint NOT CALCULATED            F-CRIT (N)

No of Constraint Violations  =   0

=============================================================================
Select optimizer option

? Evaluate_Obj_Funct                                  loop through problem code once
Option :- Evaluate_Obj_Funct


Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
```

```
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values


Checking data-base for problem definition.

*** WARNING *** - Variable U_SIGMA-B
This upper bound has no value set and has been ignored.
*** WARNING *** - Variable SIGMA-B
This variable has no constraints set and has been ignored.
*** WARNING *** - Variable U_TAU
This upper bound has no value set and has been ignored.
*** WARNING *** - Variable TAU
This variable has no constraints set and has been ignored.

Running check calculation on problem.


Re-checking data-base for problem definition.




Select optimizer option

? Show_Optimization                                   show results of first pass
Option :- Show_Optimization



================================================================================

Status after     1 evaluations is :-

                 Trial Vector

  Lwr Bound        Vector        Uppr Bound    Variable (units)

  5.000000    <  30.00000    <  50.00000      BREADTH (mm)
  2.000000    <  7.000000    <  25.00000      HEIGHT (cm)

No of V. Boundary Violations =   0

Objective Function (min.)    =  2100.000      AREA (mm^2)

             Constraints Vector

  Lwr Bound        Vector        Uppr Bound    Variable (units)
```

```
                 306.1224   <  200.0000      SIGMA-B (MPa)
                 3.571429   <  100.0000      TAU (MPa)
                 30.28237   <  5.000000      DEFLN (mm)
                 2.333333   <  10.00000      H-ON-B ()
  10000.00   <   101139.6                    F-CRIT (N)

No of Constraint Violations  =   2

==============================================================================
Select optimizer option

? Store_Data                                          update stored data
Option :- Store_Data , beam1.d
Data write to file beam1.d
File beam1.d exists, okay to overwrite.

? yes                                                 confirm existing data name

Select optimizer option

? rec beam1.d                                         recall existing data structure
Option :- Recall_Data , beam1.d
Data read from file beam1.d
Select optimizer option

? plot                                                plot contour map
Option :- Plot_Obj_Funct , Auto_Scale


Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values


Checking data-base for problem definition.


Running check calculation on problem.


Re-checking data-base for problem definition.
```

```
The objective functions is a function of BREADTH (mm) and HEIGHT (cm)
A contour map will be produced by default.
Do you wish to plot a cross-section instead

? no
*** WARNING *** - default step size for plotting both axes.



Preparing contour map of AREA (mm^2)
as a function of BREADTH (mm), from    5.000     to    50.00
versus HEIGHT (cm), from    2.000     to    25.00
consisting of  12 by   7 points.

Please WAIT.
Calculating point   1,  1 values    5.0000000    ,   2.0000000
Non-feasible point.
Calculating point   2,  1 values    9.0909090    ,   2.0000000
Non-feasible point.
Calculating point   3,  1 values    13.181818    ,   2.0000000
Non-feasible point.
Calculating point   4,  1 values    17.272728    ,   2.0000000
Non-feasible point.
Calculating point   5,  1 values    21.363636    ,   2.0000000
Non-feasible point.
Calculating point   6,  1 values    25.454546    ,   2.0000000
Non-feasible point.
Calculating point   7,  1 values    29.545454    ,   2.0000000
Non-feasible point.
.
.
.
Objective function value    7386.3633
Calculating point   8,  7 values    33.636364    ,   25.000000
Objective function value    8409.0908
Calculating point   9,  7 values    37.727272    ,   25.000000
Objective function value    9431.8184
Calculating point  10,  7 values    41.818180    ,   25.000000
Objective function value    10454.545
Calculating point  11,  7 values    45.909092    ,   25.000000
Objective function value    11477.273
Calculating point  12,  7 values    50.000000    ,   25.000000
Objective function value    12500.000

Finished calculating data points.

Display data structure

? no
Iso-metric surface

? no
Full device control

? no
Select device control option
```

*stay with 2d plot* (annotation to first "? no")

*do not show element lines* (annotation to "? no" after Display data structure)

*do not produce 3d view* (annotation to "? no" after Iso-metric surface)

*decline full graphics control at this stage* (annotation to "? no" after Full device control)

```
? sel sun                                                    choose output device
Option :- Select_Device , SUN
Output is to the SUN
Select device control option

? pi op                                                      open output device
Option :- Picture_Control , Open_Device
Select device control option

? main                                                       return to main program
Option :- Main_System
Output is to the SUN
(DEVICE OPENED)

Key box turned off.

Re-plot

? no                                                         do not re-plot the surface

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values

Select optimizer option

? plot man                                                   plot with manual scale control
Option :- Plot_Obj_Funct , Manual_Scale


(Previous output repeated      2 times.)

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values
```

```
Checking data-base for problem definition.


Running check calculation on problem.


Re-checking data-base for problem definition.





The objective functions is a function of BREADTH (mm) and HEIGHT (cm)
A contour map will be produced by default.
Do you wish to plot a cross-section instead

? no                                                    stay with 2d plot
Use raw data from WORK

? no                                      do not use previous mapping data
Input ranges for BREADTH (mm) & HEIGHT (cm) and no of pts each way

? 5/50 2/25 41/41                                        specify ranges etc.


Preparing contour map of AREA (mm^2)
as a function of BREADTH (mm), from    5.000      to    50.00
versus HEIGHT (cm), from    2.000      to    25.00
consisting of  41 by  41 points.

Please WAIT.
Calculating point   1,  1 values    5.0000000    ,    2.0000000
Non-feasible point.
Calculating point   2,  1 values    6.1250000    ,    2.0000000
Non-feasible point.
Calculating point   3,  1 values    7.2500000    ,    2.0000000
Non-feasible point.
Calculating point   4,  1 values    8.3750000    ,    2.0000000
Non-feasible point.
Calculating point   5,  1 values    9.5000000    ,    2.0000000
Non-feasible point.
Calculating point   6,  1 values    10.625000    ,    2.0000000
Non-feasible point.
Calculating point   7,  1 values    11.750000    ,    2.0000000
Non-feasible point.
.
.
.
Calculating point  33, 41 values    41.000000    ,    25.000000
Objective function value    10250.000
Calculating point  34, 41 values    42.125000    ,    25.000000
Objective function value    10531.250
Calculating point  35, 41 values    43.250000    ,    25.000000
Objective function value    10812.500
Calculating point  36, 41 values    44.375000    ,    25.000000
Objective function value    11093.750
```

```
 Calculating point  37, 41 values    45.500000     ,   25.000000
 Objective function value    11375.000
 Calculating point  38, 41 values    46.625000     ,   25.000000
 Objective function value    11656.250
 Calculating point  39, 41 values    47.750000     ,   25.000000
 Objective function value    11937.500
 Calculating point  40, 41 values    48.875000     ,   25.000000
 Objective function value    12218.750
 Calculating point  41, 41 values    50.000000     ,   25.000000
 Objective function value    12500.000

 Finished calculating data points.

(GRAPH COMPLETE)
(DEVICE CLOSED)
 Display data structure

 ? no
 Iso-metric surface

 ? no
 Full device control

 ? no
 Select device control option

 ? sel sun
 Option :- Select_Device , SUN
 Output is to the SUN
 Select device control option

 ? pi op
 Option :- Picture_Control , Open_Device
 Select device control option

 ? main
 Option :- Main_System
 Output is to the SUN
(DEVICE OPENED)

 Key box turned off.

 Re-plot

 ? no

 Resetting all "Parameter" vectors.
 Sub-option :- Set_Ctrs

 Resetting all "Variable" vectors.
 Sub-option :- Set_Ctrs

 Resetting all "Constraint" vectors.
 Sub-option :- Values

 Resetting all "Dependent" vectors.
 Sub-option :- Values
```

```
Resetting all "Tertiary" vectors.
Sub-option :- Values

Select optimizer option

? quit                                              leave system
Option :- Quit_to_System
Quit_to_System - are you sure

? yes                                               confirm quit
```

# APPENDIX G

## Terminal Interaction for Automated Example: Part I

User input follows on the lines starting with a "?". All other characters are output by the program except those in italics which are comments on the user input.

```
GENERAL PURPOSE OPTIMIZER PROGRAM


This program allows computerized optimization to be
carried out. To use the package you must provide various
subroutines and link them to this package using the "f77"
compiler. Having linked the program you can then use
the various commands available to set-up and solve your
specific problem.

If you need help try using the "help" command.



GENDAT Version A2.1 - Dated Fri Sep 22 09:28:18 1995


Data read from file opdata.d
Select optimizer option
```

? rec beam1.d *recall existing data structure*

```
Option :- Recall_Data , beam1.d
Data read from file beam1.d
Select optimizer option
```

? plot man *plot with manual scale control*

```
Option :- Plot_Obj_Funct , Manual_Scale

(Previous output repeated once.)

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs
```

```
Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values


Checking data-base for problem definition.


Running check calculation on problem.


Re-checking data-base for problem definition.

*** WARNING *** - default constraint tolerance in use, equals   0.1000E-02




The objective functions is a function of BREADTH (mm) and HEIGHT (cm)
A contour map will be produced by default.
Do you wish to plot a cross-section instead

? no                                                      stay with 2d plot
Input ranges for BREADTH (mm) & HEIGHT (cm) and no of pts each way

? 5/50 2/25 41/41                                         specify ranges etc.


Preparing contour map of AREA (mm^2)
as a function of BREADTH (mm), from    5.000     to    50.00
versus HEIGHT (cm), from    2.000     to    25.00
consisting of  41 by  41 points.

Please WAIT.
Calculating point   1,  1 values    5.0000000    ,    2.0000000
Non-feasible point.
Calculating point   2,  1 values    6.1250000    ,    2.0000000
Non-feasible point.
Calculating point   3,  1 values    7.2500000    ,    2.0000000
Non-feasible point.
Calculating point   4,  1 values    8.3750000    ,    2.0000000
Non-feasible point.
Calculating point   5,  1 values    9.5000000    ,    2.0000000
Non-feasible point.
Calculating point   6,  1 values    10.625000    ,    2.0000000
Non-feasible point.
Calculating point   7,  1 values    11.750000    ,    2.0000000
Non-feasible point.
.
.
.
```

```
Calculating point  33, 41 values    41.000000    ,   25.000000
Objective function value    10250.000
Calculating point  34, 41 values    42.125000    ,   25.000000
Objective function value    10531.250
Calculating point  35, 41 values    43.250000    ,   25.000000
Objective function value    10812.500
Calculating point  36, 41 values    44.375000    ,   25.000000
Objective function value    11093.750
Calculating point  37, 41 values    45.500000    ,   25.000000
Objective function value    11375.000
Calculating point  38, 41 values    46.625000    ,   25.000000
Objective function value    11656.250
Calculating point  39, 41 values    47.750000    ,   25.000000
Objective function value    11937.500
Calculating point  40, 41 values    48.875000    ,   25.000000
Objective function value    12218.750
Calculating point  41, 41 values    50.000000    ,   25.000000
Objective function value    12500.000

Finished calculating data points.

Display data structure

? no
Iso-metric surface

? no
Full device control

? no
Select device control option

? sel sun
Option :- Select_Device , SUN
Output is to the SUN
Select device control option

? pic op
Option :- Picture_Control , Open_Device
Select device control option

? main
Option :- Main_System
Output is to the SUN
(DEVICE OPENED)

Key box turned off.

Re-plot

? no

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs
```

```
Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values

Select optimizer option

? pl up
```
*update plot information*

```
Option :- Plot_Obj_Funct , Update

(Previous output repeated once.)

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values


Checking data-base for problem definition.


Running check calculation on problem.


Re-checking data-base for problem definition.

*** WARNING *** - default constraint tolerance in use, equals   0.1000E-02



Trace optimization path

? yes
```
*turn on optimization tracing*

```
Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values
```

```
Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values

Select optimizer option

? opt                                    select OPTIVAR optimization
Option :- OPTIVAR_Optimization , 100 , Off

(Previous output repeated once.)

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values


Checking data-base for problem definition.


Running check calculation on problem.


Re-checking data-base for problem definition.

*** WARNING *** - default constraint tolerance in use, equals   0.1000E-02



Select OPTIVAR method

? app                          use the APPROX optimization method
Option :- Approx_Optimizer , 0.1 , 46
*** WARNING *** - default stopping criteria in use, equals   0.1000E-02
*** WARNING *** - default derivative step size in use, equals   0.1000E-02

*** WARNING *** - Const. on DEFLN (mm) rounded by 0.100E-02

*** WARNING *** - Const. on H-ON-B () rounded by 0.100E-02

*** WARNING *** - Const. on DEFLN (mm) rounded by 0.100E-02

*** WARNING *** - Const. on H-ON-B () rounded by 0.100E-02

*** WARNING *** - Const. on DEFLN (mm) rounded by 0.100E-02
```

```
*** WARNING *** - Const. on H-ON-B () rounded by 0.100E-02

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values

Select optimizer option

? sho
Option :- Show_Optimization



===========================================================================

Status after     32 evaluations is :-

                    Trial Vector

   Lwr Bound        Vector        Uppr Bound    Variable (units)

  5.000000    <  15.80005    <  50.00000      BREADTH (mm)
  2.000000    <  15.80006    <  25.00000      HEIGHT (cm)

No of V. Boundary Violations =   0

Objective Function (min.)    =  2496.4165     AREA (mm^2)

                Constraints Vector

   Lwr Bound        Vector        Uppr Bound    Variable (units)

                 114.0872    <  200.0000      SIGMA-B (MPa)
                 3.004306    <  100.0000      TAU (MPa)
                 5.000010    <  5.000000      DEFLN (mm)
                 10.00001    <  10.00000      H-ON-B ()
  10000.00    <  132026.5                     F-CRIT (N)

No of Constraint Violations  =   0

===========================================================================
Select optimizer option

? in breadth 30.0                                    restore value of BREADTH
Option :- Input_Data_Value , breadth , 30.0
Select optimizer option
```

```
? in height 7.0                                       restore value of HEIGHT
Option :- Input_Data_Value , height , 7.0
Select optimizer option

? control reset          reset optimization tracing as a new map is to be produced
Option :- Control_Optimization , Reset


Objective function array subscripts and tracing reset.

Select optimizer option

? plot man
Option :- Plot_Obj_Funct , Manual_Scale


(Previous output repeated once.)

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values


Checking data-base for problem definition.


Running check calculation on problem.


Re-checking data-base for problem definition.

*** WARNING *** - default constraint tolerance in use, equals   0.1000E-02




The objective functions is a function of BREADTH (mm) and HEIGHT (cm)
A contour map will be produced by default.
Do you wish to plot a cross-section instead

? no
Use raw data from WORK

? yes                                    saves re-calculating the previous map
```

```
 Preparing contour map of AREA (mm^2)
 as a function of BREADTH (mm), from    5.000     to    50.00
 versus HEIGHT (cm), from    2.000     to    25.00
 consisting of  41 by  41 points.

 Please WAIT.

 Finished calculating data points.

(GRAPH COMPLETE)
(DEVICE CLOSED)
 Display data structure

 ? no
 Iso-metric surface

 ? no
 Full device control

 ? no
 Select device control option

 ? pic end
 Option :- Picture_Control , End_Group
 Select device control option

 ? sel sun
 Option :- Select_Device , SUN
 Output is to the SUN
 Select device control option

 ? pic op
 Option :- Picture_Control , Open_Device
 Select device control option

 ? main
 Option :- Main_System
 Output is to the PIC_TROFF
(DEVICE OPENED)

 Key box turned off.

 Re-plot

 ? no

 Resetting all "Parameter" vectors.
 Sub-option :- Set_Ctrs

 Resetting all "Variable" vectors.
 Sub-option :- Set_Ctrs

 Resetting all "Constraint" vectors.
 Sub-option :- Values

 Resetting all "Dependent" vectors.
 Sub-option :- Values
```

```
Resetting all "Tertiary" vectors.
Sub-option :- Values

Select optimizer option

? pl up
Option :- Plot_Obj_Funct , Update

(Previous output repeated once.)

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values


Checking data-base for problem definition.


Running check calculation on problem.


Re-checking data-base for problem definition.

*** WARNING *** - default constraint tolerance in use, equals   0.1000E-02



Trace optimization path

? yes

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values
```

```
Select optimizer option

? ga                                          select Genetic Algorithm optimization
Option :- GAlg_Optimization , 500 , Off


(Previous output repeated once.)

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values


Checking data-base for problem definition.


Running check calculation on problem.


Re-checking data-base for problem definition.

*** WARNING *** - default constraint tolerance in use, equals   0.1000E-02



Just highlight key trace points

? yes                                         only show best of generation on plot
Show last generation of trace points

? yes                                additionally show all final generation on plot
*** WARNING *** - default binary precision in use, equals   12
*** WARNING *** - default population size in use, equals   50
*** WARNING *** - default penalty function in use, equals   0.1000E+21
*** WARNING *** - default retention fraction in use, equals   0.8000
*** WARNING *** - default crossing fraction in use, equals   0.8000
*** WARNING *** - default inversion fraction in use, equals   0.2000
*** WARNING *** - default mutation fraction in use, equals   0.5000E-02
*** WARNING *** - default proportional survival in use.
*** WARNING *** - default clustering penalty in use, equals   0.2000
*** WARNING *** - default min. cluster sep. in use, equals   0.5000E-01
*** WARNING *** - default max. cluster size in use, equals   0.2000
*** WARNING *** - default initial no. of clusters in use, equals     10
*** WARNING *** - default min. breeding size in use, equals      5
*** WARNING *** - default random number sequence in use.
*** WARNING *** -  49 population members selected at random.
```

```
Final violated constraint penalty was   0.10000000E+21

Cluster no.   1, no. of members =  49, avg. obj. fn. val. =     0.122449E+20
minimum penalty factor =     0.157654     (factors > 1 benefit clusters)
members are  :   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
                16  17  18  19  20  21  22  23  24  25  26  27  28  29  30
                31  32  33  34  35  36  37  38  39  40  41  42  43  44  45
                46  47  48  49
centroid at  :  0.30013332      0.61480272

Cluster no.   2, no. of members =   1, avg. obj. fn. val. =     0.200000E+21
minimum penalty factor =      7.72502     (factors > 1 benefit clusters)
members are  :  50
centroid at  :  0.28260073      0.12509157

Total number of clusters is    2

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values

Select optimizer option

? sho
Option :- Show_Optimization



================================================================================

Status after    500 evaluations is :-

                    Trial Vector

   Lwr Bound        Vector        Uppr Bound    Variable (units)

  5.000000    <  16.42033     <  50.00000      BREADTH (mm)
  2.000000    <  16.37991     <  25.00000      HEIGHT (cm)

No of V. Boundary Violations =   0

Objective Function (min.)    =  2689.6360     AREA (mm^2)

               Constraints Vector

   Lwr Bound        Vector        Uppr Bound    Variable (units)
```

```
                102.1427    <  200.0000       SIGMA-B (MPa)
                2.788481    <  100.0000       TAU (MPa)
                4.318058    <  5.000000       DEFLN (mm)
                9.975387    <  10.00000       H-ON-B ()
  10000.00    < 153258.6                      F-CRIT (N)

No of Constraint Violations  =   0

==========================================================================================
Select optimizer option

? in breadth 30.0
Option :- Input_Data_Value , breadth , 30.0
Select optimizer option

? in height 7.0
Option :- Input_Data_Value , height , 7.0
Select optimizer option

? control reset
Option :- Control_Optimization , Reset


Objective function array subscripts and tracing reset.

Select optimizer option

? plot man
Option :- Plot_Obj_Funct , Manual_Scale

(Previous output repeated once.)

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values


Checking data-base for problem definition.


Running check calculation on problem.


Re-checking data-base for problem definition.

*** WARNING *** - default constraint tolerance in use, equals   0.1000E-02
```

```
The objective functions is a function of BREADTH (mm) and HEIGHT (cm)
A contour map will be produced by default.
Do you wish to plot a cross-section instead

? no
Use raw data from WORK

? yes


Preparing contour map of AREA (mm^2)
as a function of BREADTH (mm), from     5.000      to     50.00
versus HEIGHT (cm), from    2.000      to    25.00
consisting of  41 by  41 points.

Please WAIT.

Finished calculating data points.

(GRAPH COMPLETE)
(DEVICE CLOSED)
Display data structure

? no
Iso-metric surface

? no
Full device control

? no
Select device control option

? pic end
Option :- Picture_Control , End_Group
Select device control option

? sel sun
Option :- Select_Device , SUN
Output is to the SUN
Select device control option

? pic op
Option :- Picture_Control , Open_Device
Select device control option

? main
Option :- Main_System
Output is to the PIC_TROFF
(DEVICE OPENED)

Key box turned off.

Re-plot

? no

Resetting all "Parameter" vectors.
```

```
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values

Select optimizer option

? pl up
Option :- Plot_Obj_Funct , Update


(Previous output repeated once.)

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values


Checking data-base for problem definition.


Running check calculation on problem.


Re-checking data-base for problem definition.

*** WARNING *** - default constraint tolerance in use, equals   0.1000E-02



Trace optimization path

? yes

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs
```

```
Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values

Select optimizer option

? opt 1000
Option :- OPTIVAR_Optimization , 1000 , Off

(Previous output repeated once.)


Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values


Checking data-base for problem definition.


Running check calculation on problem.


Re-checking data-base for problem definition.

*** WARNING *** - default constraint tolerance in use, equals   0.1000E-02



Select OPTIVAR method

? seek fiacco
Option :- Seek_Optimizer , Fiacco-McCormick , 0.01
*** WARNING *** - default tolerance in use, equals   0.1000E-04
*** WARNING *** - default int. step size in use, equals   0.1000E-01

*** WARNING *** - Const. on H-ON-B () rounded by 0.100E-02
```

*again use OPTIVAR package but with more trials* (beside "? opt 1000")

*use Hooke and Jeeves search with Fiacco-McCormick penalty* (beside "? seek fiacco")

```
(Previous output repeated once.)

.
.
.

*** WARNING *** - Const. on DEFLN (mm) rounded by 0.100E-02

*** WARNING *** - Const. on H-ON-B () rounded by 0.100E-02


Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values

Select optimizer option

? sho
Option :- Show_Optimization



===============================================================================

Status after    773 evaluations is :-

                     Trial Vector

   Lwr Bound         Vector        Uppr Bound    Variable (units)

  5.000000    <  15.80023    <  50.00000       BREADTH (mm)
  2.000000    <  15.79922    <  25.00000       HEIGHT (cm)

No of V. Boundary Violations =   0

Objective Function (min.)     =  2496.3131      AREA (mm^2)

                Constraints Vector

   Lwr Bound         Vector        Uppr Bound    Variable (units)

                 114.0979    <  200.0000       SIGMA-B (MPa)
                 3.004431    <  100.0000       TAU (MPa)
                 5.000744    <  5.000000       DEFLN (mm)
                 9.999366    <  10.00000       H-ON-B ()
  10000.00    <  132015.7                      F-CRIT (N)
```

```
No of Constraint Violations  =   0

===============================================================================
Select optimizer option

? plot end                                    flush plot buffers and close all graphics devices
Option :- Plot_Obj_Funct , End_Plot


(Previous output repeated once.)

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values


Checking data-base for problem definition.


Running check calculation on problem.


Re-checking data-base for problem definition.

*** WARNING *** - default constraint tolerance in use, equals   0.1000E-02



(GRAPH COMPLETE)
(DEVICE CLOSED)

Resetting all "Parameter" vectors.
Sub-option :- Set_Ctrs

Resetting all "Variable" vectors.
Sub-option :- Set_Ctrs

Resetting all "Constraint" vectors.
Sub-option :- Values

Resetting all "Dependent" vectors.
Sub-option :- Values

Resetting all "Tertiary" vectors.
Sub-option :- Values
```

```
Select optimizer option

? quit                                              leave system
Option :- Quit_to_System
Quit_to_System - are you sure

? yes                                               confirm quit
```

# APPENDIX H

## Macro Command Files for Automated Example: Part II

User input for this example was prepared as two macro files which were then executed using the commands "!beam2.dbs" and "!beam2.run" (a # in the files indicates a comment line). Note that this method of operation causes the normal user prompt of a "?" to be suppressed.

**beam2.dbs**

```
rec beam1.d
in data
y
OPTIMIZER data structure - 10/12/93
This data set is used by the program "beam2" to
carry out optimization calculations on a beam.
Second version with enlarged data set and
revised objective function.





mod area ty
dep
add
cost
1 1 1
obj
pounds
Material cost.
2
add
mat_type
1 1 1
opt

Material type (1=mild steel, 2=HT steel, 3=al. alloy).
7
mod ee dim
1 1 3
mod gg dim
1 1 3
mod nu dim
1 1 3
mod sigma-y dim
1 1 3
add
rho
1 1 3
par
kg/m^3
Density.
```

```
11
add
price
1 1 3
par
pnds/t
Price per tonne.
12
add
l_mat_type
1 1 1
lim

Min. permissible material type.
29
add
u_mat_type
1 1 1
lim

Max. permissible material type.
30
add
d_mat_type
1 1 3
lim

Defined material types.
31
in mat 1.
in ee 2
210000.
72000.
in gg 2
81000.
28000.
in nu 2
0.3
0.33
in sigma-y 2
400.
395.
in rho
7850.
7850.
2800.
in price
900
1100
1400
in l_mat_type 1.0
in u_mat_type 3.0
in d_mat_type 1./1.
eval
show
sto beam2.d
print
```

```
beam2.txt
rec beam2.d
plot man
1 2
5/50 2/25 41/41
no
no
no
sel pic
pi op
main
no
plot end
#
rec beam2.d
in mat 2.
plot man
1 2
5/50 2/25 41/41
no
no
no
sel pic
pi op
main
no
plot end
#
rec beam2.d
in mat 3.
plot man
1 2
5/50 2/25 41/41
no
no
no
sel pic
pi op
main
no
plot end
#
quit
#yes
```

**beam2.run**

```
rec beam2.d
opt
app
sho
rec
ga
sho
rec
opt 1000
seek fiacco
```

```
sho
#
rec
ga
sho
opt
app
sho
#
plot man
1 3
5/50 1/3 41/41
n
n
n
sel pic
pic o
main
y
y
n
n
pic e
sel pic
pic o
main
n
mod d_mat_type ty
par
plot man
1 3
n
5/50 1/3 41/41
n
y
y
sel pic
pic o
dr
pic e
su x 75/1/15
dr
main
n
plot man
3 3
n
1 3 100
y
y
sel pic
pic o
k b l
dr
main
n
rec
```

```
in breadth 30.0
in height 7.001
opt
app
mod d_mat_type ty
par
pl m
3 3
1 3 100
y
y
sel pic
pic o
k b l
dr
pic e
main
n
q
#y
```

# APPENDIX I

## Listing for Subroutines OPTFUN and OPTCON, version II

```
      SUBROUTINE OPTFUN(ERROR)
C
C--------------------------------------------------------------------------
C
C        Routine specific type declaration block
C
      LOGICAL ERROR,DEFLT,NDEFND
      INTEGER MTYPE
      DOUBLE PRECISION GETVVD,BRDTH,HEIGHT,LENGTH,AREA,COST
      REAL GETDEF
C
C--------------------------------------------------------------------------
C
C        The main areas of routine code now follow.
C
C        n.b. length & breadth in mm and height in cm
C
      ERROR=.FALSE.
      BRDTH=MAX(1.0D-3,GETVVD('BREADTH',1,1,1,ERROR))
      IF(ERROR)GOTO 999
      HEIGHT=MAX(1.0D-3,10.0D0*GETVVD('HEIGHT',1,1,1,ERROR))
      IF(ERROR)GOTO 999
C
C        set parameters
C
      LENGTH=MAX(1.0E-3,GETDEF('LENGTH',1,1,1,1500.0,DEFLT))
C
C        decide material type
C
      MTYPE=MAX(1,MIN(3,NINT(GETVVD('MAT_TYPE',1,1,1,ERROR))))
      IF(ERROR)GOTO 999
C
C        cross-sectional area
C
      AREA=BRDTH*HEIGHT
C
C        price per tonne, dimensions in mm and density in kg/m**3 !
C
C        material 1
C
      IF(MTYPE.EQ.1)THEN
      COST=AREA*LENGTH*GETDEF('RHO',1,1,1,7850.0,DEFLT)
     1 *GETDEF('PRICE',1,1,1,900.0,DEFLT)/1.0D12
C
C        material 2
C
      ELSE IF(MTYPE.EQ.2)THEN
      COST=AREA*LENGTH*GETDEF('RHO',1,1,2,7850.0,DEFLT)
```

```
     1 *GETDEF('PRICE',1,1,2,1100.0,DEFLT)/1.0D12
C
C       material 3
C
      ELSE
      COST=AREA*LENGTH*GETDEF('RHO',1,1,3,2800.0,DEFLT)
     1 *GETDEF('PRICE',1,1,3,1400.0,DEFLT)/1.0D12
      ENDIF
C
C       save results
C
      CALL SETDFD('AREA',1,1,1,AREA,NDEFND)
      CALL SETVVD('COST',1,1,1,COST,ERROR)
      IF(ERROR)GOTO 999
C
C-----------------------------------------------------------------------
C
C       The remaining lines are standard routine exit code.
C
 999  RETURN
      END


      SUBROUTINE OPTCON(ERROR)
C
C-----------------------------------------------------------------------
C
C       Routine specific type declaration block
C
      LOGICAL ERROR,DEFLT
      INTEGER MTYPE
      DOUBLE PRECISION GETVVD,BRDTH,HEIGHT,FORCE,LENGTH,EE,GG,NU,FACTOR
     1 ,SIGMAY,SIGMAB,TAU,DEFLN,FCRIT,HONB
      REAL GETDEF
C
C-----------------------------------------------------------------------
C
C       The main areas of routine code now follow.
C
C       n.b. length & breadth in mm height in cm
C
      ERROR=.FALSE.
      BRDTH=MAX(1.0D-3,GETVVD('BREADTH',1,1,1,ERROR))
      IF(ERROR)GOTO 999
      HEIGHT=MAX(1.0D-3,10.0D0*GETVVD('HEIGHT',1,1,1,ERROR))
      IF(ERROR)GOTO 999
C
C       set parameters
C
      LENGTH=MAX(1.0E-3,GETDEF('LENGTH',1,1,1,1500.0,DEFLT))
      FORCE=GETDEF('FORCE',1,1,1,5000.0,DEFLT)
      FACTOR=GETDEF('FACTOR',1,1,1,2.0,DEFLT)
C
C       decide material type and set material properties
C
      MTYPE=MAX(1,MIN(3,NINT(GETVVD('MAT_TYPE',1,1,1,ERROR))))
      IF(ERROR)GOTO 999
```

```
C
C         material 1
C
      IF(MTYPE.EQ.1)THEN
      EE=GETDEF('EE',1,1,1,216620.0,DEFLT)
      GG=GETDEF('GG',1,1,1,86650.0,DEFLT)
      NU=GETDEF('NU',1,1,1,0.27,DEFLT)
      SIGMAY=GETDEF('SIGMAY',1,1,1,200.0,DEFLT)
C
C         material 2
C
      ELSE IF(MTYPE.EQ.2)THEN
      EE=GETDEF('EE',1,1,2,210000.0,DEFLT)
      GG=GETDEF('GG',1,1,2,81000.0,DEFLT)
      NU=GETDEF('NU',1,1,2,0.30,DEFLT)
      SIGMAY=GETDEF('SIGMAY',1,1,2,400.0,DEFLT)
C
C         material 3
C
      ELSE
      EE=GETDEF('EE',1,1,3,72000.0,DEFLT)
      GG=GETDEF('GG',1,1,3,28000.0,DEFLT)
      NU=GETDEF('NU',1,1,3,0.33,DEFLT)
      SIGMAY=GETDEF('SIGMAY',1,1,3,395.0,DEFLT)
      ENDIF
C
C         set material limits (tau-y = sigma-y / 2)
C
      CALL SETVVD('U_SIGMA-B',1,1,1,SIGMAY,ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('U_TAU',1,1,1,SIGMAY/2.0D0,ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('L_F-CRIT',1,1,1,FORCE*FACTOR,ERROR)
      IF(ERROR)GOTO 999
C
C         bending stress
C
      SIGMAB=6.0*FORCE*LENGTH/(BRDTH*HEIGHT*HEIGHT)
C
C         shear stress
C
      TAU=1.5*FORCE/(BRDTH*HEIGHT)
C
C         tip deflection
C
      DEFLN=4.0*FORCE*LENGTH**3/(EE*BRDTH*HEIGHT**3)
C
C         twist buckling instability load
C
      FCRIT=(4.0*SQRT(GG*(BRDTH**3*HEIGHT+HEIGHT**3
     1 *BRDTH)*EE*(BRDTH**3*HEIGHT)
     2 /(144.0*(1.0-NU*NU)))/(LENGTH*LENGTH))
C
C         section aspect ratio
C
      HONB=HEIGHT/BRDTH
C
```

```
C         save results, limiting them to sensible numbers
C
      CALL SETVVD('SIGMA-B',1,1,1,MIN(1.0D15,SIGMAB),ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('TAU',1,1,1,MIN(1.0D15,TAU),ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('DEFLN',1,1,1,MIN(1.0D15,DEFLN),ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('F-CRIT',1,1,1,MIN(1.0D15,FCRIT),ERROR)
      IF(ERROR)GOTO 999
      CALL SETVVD('H-ON-B',1,1,1,MIN(1.0D15,HONB),ERROR)
      IF(ERROR)GOTO 999
C
C-----------------------------------------------------------------------
C
C         The remaining lines are standard routine exit code.
C
  999 RETURN
      END
```

# APPENDIX J

## Typical Advanced Listings for Subroutine OPTFUN

Fortran invocation using OPTCHK to test if calls to a particular routine have calculated the variable currently flagged as being the objective function in the GENDAT data-base. This allows a long, time-consuming series of subroutine calls to be linked in during compilation but then only those needed at run time are actually called.

```
      SUBROUTINE OPTFUN(ERROR)
C
C-----------------------------------------------------------------------
C
C
C        ==================================
C        COPYRIGHT   A.J. KEANE   13 DEC 86.
C        ==================================
C
C
C        PART OF :-
C        ==========
C
C          This routine is an example design master optimization routine.
C
C        PURPOSE :-
C        ==========
C
C          This routine is used to evaluate a user defined objective function
C          or user defined constraint vector applying to the design being
C          handled.
C
C        METHOD :-
C        =========
C
C          This routine evaluates the designer chosen objective function
C          by running through the design routines in the normal sequence and
C          checking at each stage if the objective function has been calculated.
C          When it has it then goes no further through the design, but instead
C          just returns. This minimizes the calculation required while still
C          allowing any variable to form the objective. It also allows the
C          design modules to be changed in number and sequence with little
C          effort.
C
C          Because this method is equally applicable to calculating constraints
C          this routine has an alternate ENTRY point called OPTCON which uses
C          the same sequence of calculation modules, but instead checks that
C          ALL the constraints are calculated before returning.
C
C          Both entries return an with ERROR set .TRUE. if all the calculation
C          modules are run through and the relevant functions have not been
C          calculated. This also occurs if any of the calculation modules fail.
C
C        NON-STANDARD :-
```

```
C        ===============
C
C          This routine has NO non-standard elements.
C
C        RESTRICTIONS :-
C        ===============
C
C          none
C
C        COMMUNICATION :-
C        ================
C
C          ARGUMENTS      - Input args and meanings :-
C                           none
C                           Input/Output args and meanings :-
C                           none
C                           Output args and meanings :-
C                           ERROR  - The error flag. A LOGICAL variable.
C          COMMON BLOCKS - Common Blocks used internally and modifications :-
C                           CPRTSW - The print switch block, which is checked to
C                                    see if debug output is required.
C                           CPRTNM - The print switch names block, which is
C                                    checked against routine names to
C                                    see if debug output is required.
C                           IOLUN  - The I/O block, which sets the logical unit
C                                    numbers for terminal input and output and
C                                    holds a logical flag which indicates if
C                                    any routine has detected a mistake in user
C                                    input.
C                           CWSTRG - The write string block, which contains the
C                                    characters to be output in the CHARACTER*256
C                                    array OUTSTR.
C                                    see if debug output is required.
C                           IOLUN  - The I/O block, which sets the logical unit
C                                    numbers for terminal input and output.
C                           OPTMZ1/2 - Pass details on the LUNs, control
C                                    parameters, names and positions of data
C                                    in the GENDAT package.
C          FILES          - Files used and purposes :-
C                           none
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
C        COMMON BLOCKS :-
C        ================
C
C          The following Common Blocks are used in this routine :-
C
C            CPRTSW - This block is used to pass the debugging print switch,
C                     the logical unit to which output is to be sent and the
C                     level of output (full or not).
C            CPRTNM - This block is used to pass the names of routines for which
C                     debugging output is required.
C            IOLUN  - This block is used to set the LUNs for terminal I/O.
C                     It is not required if input, output or checking of user
C                     responses does not take place within this routine.
C            CWSTRG - This block is used to pass the output characters in the
```

```
C                 CHARACTER*256 array OUTSTR.
C          GDIN01 - This block is used to hold the package initialisation
C                   string.
C          DAT001 - This block holds data handler information.
C          DAT002 - This block holds data handler information.
C          CPRTSW - This block is used to pass the debugging print switch,
C                   the LUN to which output is to be sent and the level of
C                   output (full or not).
C          IOLUN  - This block is used to set the LUNs for terminal I/O.
C                   It is not required if input and output do not take place
C                   within this routine.
C          OPTMZ1/2 - Pass details on the LUNs, control parameters and
C                   positions of data in the GENDAT package. In particular
C                   it shows which stack values are for design variables,
C                   constraints and ranges; allowing rapid GENDAT access
C                   without using the get and set routines.
C                   Also contains name and units strings for output.
C
C      INTERNAL VARIABLES :-
C      =====================
C
C        The following internal variables are used in this routine :-
C
C          CFLAG  - set true when constraints rather than the objective
C                   function are to be calculated.
C          CLOOP  - constraint loop counter.
C          FLOOP  - objective function loop counter.
C          DEFLT  - set true if a default is returned for the loop counters.
C          PSFLAG - set done if we have a set of variable position pointers.
C          OPTYPE - the type of variable relevant to the entry point used.
C          CTYPE  - variable type returned by GETVI.
C          C      - dummy character variable used by GETVI.
C          I      - dummy integer variable used by GETVI.
C          DONE   - set true when the required calculations have been done.
C
C      PRINT SWITCH :-
C      ===============
C
C        The LOGICAL variable PRTSW, which is passed in the common block
C        CPRTSW, is used as a print switch. When it is .TRUE. a number of
C        program debugging lines are turned on. These output messages on
C        the logical unit defined by LOGLUN. If the debugging level set by the
C        common integer variable LEVEL is greater than or equal to the routine
C        level set by the common integer variable PLEVEL, then the complete
C        details of arguments passed to the routine and back, are output,
C        whilst if it is one less only the routine name and direction are
C        given. Otherwise output is only given if the routine name occurs in
C        the list of names held in the CHARACTER*6 array PNAMES held in the
C        block CPRTNM. PLEVEL is incremented on entry and decremented on exit.
C        The flag PFULL is used to indicate that all routines should output
C        if required and is checked by library routines prior to output.
C
C      SUBROUTINES CALLED :-
C      =====================
C
C        The following subroutines and functions are used by this routine :-
C
```

```
C             design software modules.
C             GENDAT package to handler data transfers.
C
C         CODING :-
C         =========
C
C           STANDARD ADHERED TO :-  ANSI FORTRAN 77
C           CODED BY            :-  A.J. KEANE
C
C         REVISION :-
C         ===========
C
C           DATE        :- 15/12/86
C           VERSION     :- A1.0
C           DESCRIPTION :- First version, revised print switch and COMMON LUNs.
C
C-----------------------------------------------------------------------
C
C         The following area of code defines the general purpose debugging print
C         switch, I/O logical unit numbers, input checking flag and the output
C         string.
C
      LOGICAL PRTSW,PFULL,PMUST
      INTEGER LOGLUN,LEVEL,PLEVEL,PCOUNT
      COMMON/CPRTSW/PRTSW,PFULL,LOGLUN,LEVEL,PLEVEL,PCOUNT
      SAVE /CPRTSW/
      CHARACTER*6 PNAMES(10)
      COMMON/CPRTNM/PNAMES
      SAVE /CPRTNM/
      LOGICAL IOCHCK
      INTEGER INLUN,OUTLUN
      COMMON/IOLUN/INLUN,OUTLUN,IOCHCK
      SAVE /IOLUN/
      CHARACTER*256 OUTSTR(50)
      COMMON/CWSTRG/OUTSTR
      SAVE /CWSTRG/
      EXTERNAL IODAT
C
C-----------------------------------------------------------------------
C
C         Routine specific type declaration block
C
C         optimizer common blocks
C
      INTEGER MAXVAR,MAXCST
      PARAMETER(MAXVAR=200,MAXCST=200)
      LOGICAL BOUNDS
      INTEGER ILUN,OLUN,GENLEV,MONFRQ,MNITER,NOVARS,NCNSTR,NEQ,NINEQ
     1 ,NRANGE,OBJPNT,OBJPOS,OBJXYZ(3),VARPNT(MAXVAR),VARPOS(MAXVAR)
     2 ,LVRPNT(MAXVAR),UVRPNT(MAXVAR),NDVARS(MAXVAR),DVRPNT(MAXVAR)
     3 ,CSTPNT(MAXCST),CSTPOS(MAXCST),LCTPNT(MAXCST),UCTPNT(MAXCST)
     4 ,ECTPNT(MAXCST),ORSPNT(3),OPTCTR(2*MAXCST+1)
      REAL CTOLER
      CHARACTER*30 OBJNAM,VARNAM(MAXVAR),CSTNAM(MAXCST)
      CHARACTER*256 PRGNAM
      COMMON/OPTMZ1/BOUNDS,ILUN,OLUN,GENLEV,MONFRQ,MNITER,NOVARS
     1 ,NCNSTR,NEQ,NINEQ,NRANGE,OBJPNT,OBJPOS,OBJXYZ,VARPNT,VARPOS
```

```
     2 ,LVRPNT,UVRPNT,NDVARS,DVRPNT,CSTPNT,CSTPOS,LCTPNT,UCTPNT
     3 ,ECTPNT,ORSPNT,OPTCTR,CTOLER
      COMMON/OPTMZ2/OBJNAM,VARNAM,CSTNAM,PRGNAM
      SAVE /OPTMZ1/,/OPTMZ2/
C
      LOGICAL ERROR,CFLAG,DEFLT,PSFLAG,DONE
      INTEGER CLOOP,FLOOP,I
      CHARACTER C
      CHARACTER*10 OPTYPE,CTYPE
C
C-----------------------------------------------------------------------
C
C     Alternate entry point mechanism
C
      CFLAG=.FALSE.
      GOTO 10
      ENTRY OPTCON(ERROR)
      CFLAG=.TRUE.
 10   CONTINUE
C
C-----------------------------------------------------------------------
C
C     The next few lines are standard routine entry code.
C
      PLEVEL=PLEVEL+1
      IF(PRTSW)THEN
      PMUST=.FALSE.
      DO 99999 PCOUNT=1,10
      IF((CFLAG.AND.PNAMES(PCOUNT).EQ.'OPTCON')
     1 .OR.(.NOT.CFLAG.AND.PNAMES(PCOUNT).EQ.'OPTFUN'))PMUST=.TRUE.
99999 CONTINUE
      IF(PMUST.OR.LEVEL.GE.PLEVEL-1)THEN
      IF(PMUST.OR.LEVEL.GE.PLEVEL)THEN
      WRITE(OUTSTR,9990)
 9990 FORMAT(1X,75('-'))
      CALL WSTRNG(LOGLUN,0)
      ENDIF
      IF(CFLAG)THEN
      WRITE(OUTSTR,9991)'OPTCON'
 9991 FORMAT(1X,'Routine ',A6,' - Incoming')
      CALL WSTRNG(LOGLUN,0)
      ELSE
      WRITE(OUTSTR,9991)'OPTFUN'
      CALL WSTRNG(LOGLUN,0)
      ENDIF
      IF(PMUST.OR.LEVEL.GE.PLEVEL)THEN
      WRITE(OUTSTR,9992)
 9992 FORMAT(1X,'Input arguments :-'/
     1       1X,'none')
      CALL WSTRNG(LOGLUN,0)
      WRITE(OUTSTR,9993)
 9993 FORMAT(1X,'Input/Output arguments :-'/
     1       1X,'none')
      CALL WSTRNG(LOGLUN,0)
      WRITE(OUTSTR,9990)
      CALL WSTRNG(LOGLUN,0)
      ENDIF
```

```
      ENDIF
      ENDIF
C
C-----------------------------------------------------------------------
C
C        The main areas of routine code now follow.
C
C        first increment loop counters if these are defined in the data-base
C
      ERROR=.FALSE.
      IF(CFLAG)THEN
      CLOOP=NINT(GETDEF('C_LOOPS',1,1,1,0.0,DEFLT))
      IF(.NOT.DEFLT)CALL SETVV('C_LOOPS',1,1,1,REAL(CLOOP+1),ERROR)
      ELSE
      FLOOP=NINT(GETDEF('F_LOOPS',1,1,1,0.0,DEFLT))
      IF(.NOT.DEFLT)CALL SETVV('F_LOOPS',1,1,1,REAL(FLOOP+1),ERROR)
      ENDIF
      IF(ERROR)GOTO 999
C
C        now set type and save and clear appropriate set counter
C
      IF(CFLAG)THEN
      OPTYPE='Constraint'
      ELSE
      OPTYPE='Obj_Funct'
      ENDIF
      CALL OPTCHK(OPTYPE,'Clear',PSFLAG,ERROR)
      IF(ERROR.AND.PSFLAG)THEN
      ERROR=.FALSE.
      PSFLAG=.FALSE.
      ELSE IF(ERROR.AND..NOT.PSFLAG)THEN
      GOTO 999
      ENDIF
      IF(PSFLAG)THEN
      CALL OPTCHK(OPTYPE,'Check',DONE,ERROR)
      IF(ERROR)GOTO 999
      IF(DONE)GOTO 990
      ENDIF
C
C        first design module
C
      CALL DSGN1(ERROR)
      IF(ERROR)GOTO 990
      CALL NDSPCH
      IF(PSFLAG)THEN
      CALL OPTCHK(OPTYPE,'Check',DONE,ERROR)
      IF(ERROR)GOTO 999
      IF(DONE)GOTO 990
      ENDIF
C
C        second design module
C
      CALL DSGN2(ERROR)
      IF(ERROR)GOTO 990
      CALL NDSPCH
      IF(PSFLAG)THEN
      CALL OPTCHK(OPTYPE,'Check',DONE,ERROR)
```

```
      IF(ERROR)GOTO 999
      IF(DONE)GOTO 990
      ENDIF
C
C        third design module
C
      CALL DSGN3(ERROR)
      IF(ERROR)GOTO 990
      CALL NDSPCH
      IF(PSFLAG)THEN
      CALL OPTCHK(OPTYPE,'Check',DONE,ERROR)
      IF(ERROR)GOTO 999
      IF(DONE)GOTO 990
      ENDIF
C
C        fourth design module
C
      CALL DSGN4(ERROR)
      IF(ERROR)GOTO 990
      CALL NDSPCH
      IF(PSFLAG)THEN
      CALL OPTCHK(OPTYPE,'Check',DONE,ERROR)
      IF(ERROR)GOTO 999
      IF(DONE)GOTO 990
      ENDIF
C
C       if we had a set of pointer flags and get here we have not satisfied
C       the required calculation but have used all the available design
C       modules so this is an error
C
      IF(PSFLAG)ERROR=.TRUE.
C
C        now reset appropriate counters
C
 990  IF(PSFLAG)THEN
      IF(ERROR)THEN
      CALL OPTCHK(OPTYPE,'Restore',DONE,ERROR)
      ERROR=.TRUE.
      ELSE
      CALL OPTCHK(OPTYPE,'Restore',DONE,ERROR)
      IF(ERROR.AND.DONE)ERROR=.FALSE.
      ENDIF
      ENDIF
C
C-----------------------------------------------------------------------
C
C        The remaining lines are standard routine exit code.
C
 999  IF(PRTSW)THEN
      IF(PMUST.OR.LEVEL.GE.PLEVEL-1)THEN
      IF(PMUST.OR.LEVEL.GE.PLEVEL)THEN
      WRITE(OUTSTR,9990)
      CALL WSTRNG(LOGLUN,0)
      ENDIF
      IF(CFLAG)THEN
      WRITE(OUTSTR,9994)'OPTCON'
 9994 FORMAT(1X,'Routine ',A6,' - Outgoing')
```

```
      CALL WSTRNG(LOGLUN,0)
      ELSE
      WRITE(OUTSTR,9994)'OPTFUN'
      CALL WSTRNG(LOGLUN,0)
      ENDIF
      IF(PMUST.OR.LEVEL.GE.PLEVEL)THEN
      WRITE(OUTSTR,9992)
      CALL WSTRNG(LOGLUN,0)
      WRITE(OUTSTR,9993)
      CALL WSTRNG(LOGLUN,0)
      WRITE(OUTSTR,9995)ERROR
 9995 FORMAT(1X,'Output arguments :-'/
     1       1X,'ERROR  = ',L4)
      CALL WSTRNG(LOGLUN,0)
      WRITE(OUTSTR,9990)
      CALL WSTRNG(LOGLUN,0)
      ENDIF
      ENDIF
      ENDIF
      PLEVEL=PLEVEL-1
      RETURN
      END
```

C invocation illustrating how the various GENDAT routines are called. Note that it is assumed the final linking of the code will be done using the fortran compiler together with the pre-compiled C objects.

```
#include <stdio.h>
#include <string.h>

static FILE *fp = NULL;

/* Define the maximum number of variables we will use */

#define MXVARS                                                  120

/* Table of constant values that we can refer to by pointer */

static long int false = 0;
static long int one = 1;

/* Simple definition of min and max functions */

#ifndef max
#define max(a,b)                        (((a) > (b)) ? (a) : (b))
#endif
#ifndef min
#define min(a,b)                        (((a) < (b)) ? (a) : (b))
#endif

/* Simple exponentiation routine to return powers of 2 */

long int power2(nn)
long int nn;
{
    int i, n;
    long int pp;
    pp = 1L;
    n = nn;
```

```
    for (i = 1; i <=n; ++i)
                                                pp = pp * 2L;
    return(pp);
}

/* Simple NINT routine to convert +ve floats to nearest integer */

long int nint(rr)
float rr;
{
    long int idummy1, idummy2;
    idummy1 = rr;
    idummy2 = rr+0.5;
    if ( idummy1 == idummy2 ) {
                                                return(idummy1);
    } else {
                                                return(idummy2);
    }
}

/* Subroutine */ int optfun_(error)
long int *error;
{
    /* Local variables */
    static long int nvars, ii, jj, icont[8], jcont[MXVARS], nvon;
    static float vars[MXVARS], fit;
    static char string[2048];
    static char sdummy[15];
    extern /* Function */ float getvv_();
    extern /* Subroutine */ int setvv_();
    extern /* Subroutine */ int gfun_();
    extern /* Subroutine */ int packwd_();

/* The main areas of routine code now follow. */

/* get data */

    *error = false;
    nvars = min(MXVARS, nint(getvv_("NVARS", &one, &one, &one, error, 5L)));
    if (*error) goto L999;
    for (ii = 1; ii <= min(8, nvars / 12 + 1); ++ii) {
1] = min(4095,max(0, nint(getvv_("CONTROL", &one, &one, &ii, error, 7L))));
                                    if (*error) goto L999;
                              for (jj = 12; jj >= 1; --jj) {
                        if (icont[ii - 1] >= power2(jj - 1)) {
                                                      jcont[(ii - 1) *
                                                      icont[ii - 1] -=
                                          } else {
                                                      jcont[(ii - 1) *
                                                      }
                                                      }
    }

    for (ii = 1; ii <= nvars; ++ii) {
                                    if (jcont[ii - 1] != 0) {
      vars[ii - 1] = getvv_("VARS", &one, &one, &ii, error, 4L);
                                    if (*error) goto L999;
```

```
                                                        } else {
                                              vars[ii - 1] = 0.0;
                                                                }
    }
    strcpy(string, "");
    for (ii = 1; ii <= nvars; ++ii) {
                        sprintf(sdummy, " %10.7f", vars[ii - 1]);
                                            strcat(string, sdummy);
    }
    ii = strlen(string);
    packwd_(&false, string, &jj, ii);
    string[jj] = ' ';

/* calculate results */

/* pass string to gfun to calculate fitness */

#ifdef DEBUG
printf("%s0, string);
#endif

    gfun_(string, &fit, jj);

#ifdef DEBUG
printf("%f0, fit);
#endif

/* save results */

    setvv_("FITNESS", &one, &one, &one, &fit, error, 7L);
    if (*error) goto L999;
    nvon = 0;
    for (ii = 1; ii <= nvars; ++ii) {
                                          if (jcont[ii - 1] != 0) {
                                                          ++nvon;
                                                                }
    }

/* set up string info for storing to a file for logging */

    strcpy(string, "");
    sprintf(sdummy, "%6.3f %3d ", fit, nvon);
    strcat(string, sdummy);
    for (ii = 1; ii <= nvars; ++ii) {

/* if you uncomment the next "if" sequence instead of the following two lines
   you only get the active variables printed */
                                          if (jcont[ii - 1] != 0) {
                 sprintf(sdummy, " %2d %10.7f", ii, vars[ii - 1]);
                                            strcat(string, sdummy);
                                                                }
/*                      sprintf(sdummy, " %10.7f", vars[ii - 1]);*/
/*                                          strcat(string, sdummy);*/
    }
    ii = strlen(string);
    packwd_(&false, string, &jj, ii);
    string[jj] = ' ';
```

```
    if ( fp == NULL ) fp = fopen("fort.77_c", "w");
    fprintf(fp, "%s0, string);

L999:
    return 0;
} /* optfun_ */
```

# APPENDIX K

## Listing for programs beam1s.f and beam2s.f

```
C         PROGRAM BEAM1S
          SUBROUTINE WINMNU_PROGRAM(OPTPRG)
C
C           example program that uses the OPTIONS subroutine interface to run the
C           first example in the manual
C
          INTEGER NVRS,NDVRS,NCONS,NPARAM,OLEVEL
          DOUBLE PRECISION DNULL
          PARAMETER(NVRS=2,NDVRS=1,NCONS=5,NPARAM=7,OLEVEL=1,DNULL=-777.0D0)
          LOGICAL ERROR
          INTEGER NITERS,DIRCTN,NCALLS
          DOUBLE PRECISION VARS(NVRS),LVARS(NVRS),UVARS(NVRS)
         1  ,DVARS(NVRS,NDVRS),CONS(NCONS),LCONS(NCONS),UCONS(NCONS)
         2  ,PARAMS(NPARAM),OBJFN,OMETHD
          CHARACTER*(*) OPTPRG
          CHARACTER*10 VARNAM(NVRS),CONNAM(NCONS),PARNAM(NPARAM),OBJNAM
C
C           define the design vector, constraints and objective function
C
C           design vector limits (n.b., to unset a limit set it equal to DNULL)
C
          LVARS(1)=5.0D0
          LVARS(2)=2.0D0
          UVARS(1)=50.0D0
          UVARS(2)=25.0D0
          DVARS(1,1)=DNULL
          DVARS(2,1)=DNULL
          VARNAM(1)='BREADTH'
          VARNAM(2)='HEIGHT'
C
C           constraints vector limits (n.b., to unset a limit set it equal to DNULL)
C
          LCONS(1)=DNULL
          LCONS(2)=DNULL
          LCONS(3)=DNULL
          LCONS(4)=DNULL
          LCONS(5)=5000.0D0
          UCONS(1)=200.0
          UCONS(2)=100.0
          UCONS(3)=5.0D0
          UCONS(4)=10.0D0
          UCONS(5)=DNULL
          CONNAM(1)='SIGMA-B'
          CONNAM(2)='TAU'
          CONNAM(3)='DEFLN'
          CONNAM(4)='H-ON-B'
          CONNAM(5)='F-CRIT'
C
```

```
C          parameters
C
      PARNAM(1)='LENGTH'
      PARNAM(2)='FORCE'
      PARNAM(3)='FACTOR'
      PARNAM(4)='EE'
      PARNAM(5)='GG'
      PARNAM(6)='NU'
      PARNAM(7)='SIGMAY'
      PARAMS(1)=1500.0D0
      PARAMS(2)=5000.0D0
      PARAMS(3)=2.0D0
      PARAMS(4)=216620.0D0
      PARAMS(5)=86650.0D0
      PARAMS(6)=0.27D0
      PARAMS(7)=200.0D0
C
C          objective function
C
      OBJNAM='AREA'
C
C          set up the structure of the problem
C
      PRINT *,'Calling OPTDBS with arguments of '
     1  ,OLEVEL,DNULL,NVRS,NDVRS,NCONS,NPARAM,VARNAM
     2  ,CONNAM,PARNAM,OBJNAM,LVARS,UVARS,DVARS,LCONS,UCONS,PARAMS
      CALL OPTDBS(OLEVEL,DNULL,NVRS,NDVRS,NCONS,NPARAM,VARNAM
     1  ,CONNAM,PARNAM,OBJNAM,LVARS,UVARS,DVARS,LCONS,UCONS,PARAMS,ERROR)
      PRINT *,'Returned from OPTDBS with results of '
     1  ,OLEVEL,DNULL,NVRS,NDVRS,NCONS,NPARAM,VARNAM
     2  ,CONNAM,PARNAM,OBJNAM,LVARS,UVARS,DVARS,LCONS,UCONS,PARAMS,ERROR
      IF(ERROR)THEN
      PRINT *,'Unable to define the problem vectors'
      STOP
      ENDIF
C     CALL OPTCTL
C
C          search the problem using a given method for a given number of steps
C          (method 1.8 is an OPTIVAR Approx search)
C
      OMETHD=1.8
      VARS(1)=30.0
      VARS(2)=7.0
      DIRCTN=-1
      NITERS=100
      PRINT *,' '
      PRINT *,'Calling OPTNS with arguments of '
     1  ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1  ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1  ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to search the problem'
      STOP
      ENDIF
C
```

```
C        search the problem using a given method for a given number of steps
C        (method 4.0 is a GA search)
C
      OMETHD=4.0
      VARS(1)=30.0
      VARS(2)=7.0
      DIRCTN=-1
      NITERS=500
      PRINT *,' '
      PRINT *,'Calling OPTNS with arguments of '
     1 ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1 ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1 ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to search the problem'
      STOP
      ENDIF
C
C        search the problem using a given method for a given number of steps
C        (method 1.6 is an OPTIVAR Seek search)
C
      OMETHD=1.6
      VARS(1)=30.0
      VARS(2)=7.0
      DIRCTN=-1
      NITERS=1000
C
C        set the Fiacco-McCormick penalty
C
      ERROR=.FALSE.
      CALL SETVV('OVR_PENAL',1,1,1,2.0,ERROR)
      IF(ERROR)THEN
      PRINT *,'Failed to set search parameter OVR_PENAL'
      STOP
      ENDIF
      PRINT *,' '
      PRINT *,'Calling OPTNS with arguments of '
     1 ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1 ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1 ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to search the problem'
      STOP
      ENDIF
C
C        just evaluate the problem
C
      OMETHD=0.0
      VARS(1)=30.0
      VARS(2)=7.0
      DIRCTN=-1
      NITERS=1
      PRINT *,' '
```

```
      PRINT *,'Calling OPTNS with arguments of '
     1 ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1 ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1 ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to evaluate the problem'
      STOP
      ENDIF
C
      STOP
      END


C       PROGRAM BEAM2S
      SUBROUTINE WINMNU_PROGRAM(OPTPRG)
C
C        example program that uses the OPTIONS subroutine interface to run the
C        second example in the manual
C
      INTEGER NVRS,NDVRS,NCONS,NPARAM,OLEVEL
      DOUBLE PRECISION DNULL
      PARAMETER(NVRS=3,NDVRS=3,NCONS=5,NPARAM=1,OLEVEL=1,DNULL=-777.0D0)
      LOGICAL ERROR
      INTEGER NITERS,DIRCTN,NCALLS
      DOUBLE PRECISION VARS(NVRS),LVARS(NVRS),UVARS(NVRS)
     1 ,DVARS(NVRS,NDVRS),CONS(NCONS),LCONS(NCONS),UCONS(NCONS)
     2 ,PARAMS(NPARAM),OBJFN,OMETHD
      CHARACTER*(*) OPTPRG
      CHARACTER*10 VARNAM(NVRS),CONNAM(NCONS),PARNAM(NPARAM),OBJNAM
C
C        define the design vector, constraints and objective function
C
C        design vector limits (n.b., to unset a limit set it equal to DNULL)
C
      LVARS(1)=5.0D0
      LVARS(2)=2.0D0
      LVARS(3)=1.0D0
      UVARS(1)=50.0D0
      UVARS(2)=25.0D0
      UVARS(3)=3.0D0
      DVARS(1,1)=DNULL
      DVARS(1,2)=DNULL
      DVARS(1,3)=DNULL
      DVARS(2,1)=DNULL
      DVARS(2,2)=DNULL
      DVARS(2,3)=DNULL
      DVARS(3,1)=1.0D0
      DVARS(3,2)=2.0D0
      DVARS(3,3)=3.0D0
      VARNAM(1)='BREADTH'
      VARNAM(2)='HEIGHT'
      VARNAM(3)='MAT_TYPE'
C
C        constraints vector limits (n.b., to unset a limit set it equal to DNULL)
C
```

```
      LCONS(1)=DNULL
      LCONS(2)=DNULL
      LCONS(3)=DNULL
      LCONS(4)=DNULL
      LCONS(5)=5000.0D0
      UCONS(1)=200.0
      UCONS(2)=100.0
      UCONS(3)=5.0D0
      UCONS(4)=10.0D0
      UCONS(5)=DNULL
      CONNAM(1)='SIGMA-B'
      CONNAM(2)='TAU'
      CONNAM(3)='DEFLN'
      CONNAM(4)='H-ON-B'
      CONNAM(5)='F-CRIT'
C
C       parameters (none in this example - instead rely on GETDEF defaults)
C
      PARAMS(1)=DNULL
      PARNAM(1)=' '
C
C       objective function
C
      OBJNAM='COST'
C
C       set up the structure of the problem
C
      CALL OPTDBS(OLEVEL,DNULL,NVRS,NDVRS,NCONS,NPARAM,VARNAM
     1 ,CONNAM,PARNAM,OBJNAM,LVARS,UVARS,DVARS,LCONS,UCONS,PARAMS,ERROR)
      IF(ERROR)THEN
      PRINT *,'Unable to define the problem vectors'
      STOP
      ENDIF
C     CALL OPTCTL
C
C       search the problem using a given method for a given number of steps
C       (method 1.8 is an OPTIVAR Approx search)
C
      OMETHD=1.8
      VARS(1)=30.0
      VARS(2)=7.0
      VARS(3)=1.0
      DIRCTN=-1
      NITERS=100
      PRINT *,' '
      PRINT *,'Calling OPTNS with arguments of '
     1 ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1 ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1 ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to search the problem'
      STOP
      ENDIF
C
C       search the problem using a given method for a given number of steps
```

```
C         (method 4.0 is a GA search)
C
      OMETHD=4.0
      VARS(1)=30.0
      VARS(2)=7.0
      VARS(3)=1.0
      DIRCTN=-1
      NITERS=500
      PRINT *,' '
      PRINT *,'Calling OPTNS with arguments of '
     1 ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1 ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1 ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to search the problem'
      STOP
      ENDIF
C
C         search the problem using a given method for a given number of steps
C         (method 1.6 is an OPTIVAR Seek search)
C
      OMETHD=1.6
      VARS(1)=30.0
      VARS(2)=7.0
      VARS(3)=1.0
      DIRCTN=-1
      NITERS=1000
C
C         set the Fiacco-McCormick penalty
C
      ERROR=.FALSE.
      CALL SETVV('OVR_PENAL',1,1,1,2.0,ERROR)
      IF(ERROR)THEN
      PRINT *,'Failed to set search parameter OVR_PENAL'
      STOP
      ENDIF
      PRINT *,' '
      PRINT *,'Calling OPTNS with arguments of '
     1 ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1 ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1 ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to search the problem'
      STOP
      ENDIF
C
C         search the problem using a given method for a given number of steps
C         (method 4.0 is a GA search)
C
      OMETHD=4.0
      VARS(1)=30.0
      VARS(2)=7.0
      VARS(3)=1.0
```

```
      DIRCTN=-1
      NITERS=500
      PRINT *,' '
      PRINT *,'Calling OPTNS with arguments of '
     1 ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1 ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1 ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to search the problem'
      STOP
      ENDIF
C
C        search the problem using a given method for a given number of steps
C        but do not reset the starting position
C        (method 1.8 is an OPTIVAR Approx search)
C
      OMETHD=1.8
      DIRCTN=-1
      NITERS=100
C
C        reset the penalty to the internal default
C
      ERROR=.FALSE.
      CALL SETVV('OVR_PENAL',-1,-1,-1,0.0,ERROR)
      IF(ERROR)THEN
      PRINT *,'Failed to reset search parameter OVR_PENAL'
      STOP
      ENDIF
      PRINT *,' '
      PRINT *,'Calling OPTNS with arguments of '
     1 ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1 ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1 ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to search the problem'
      STOP
      ENDIF
C
      STOP
      END
```

# APPENDIX L

## Usage and Default Values for System Variables

This appendix list the routines that use system variables, their names and default values. Their meanings and applications are described in Chapter 8.

| Function | System Variable | Default | Notes |
|---|---|---|---|
| all functions | OPT_CTOL | 1.0E-3 | |
| | OPT_TRACE | none | used to store objective function calculation details |
| | OPT_CTRACE | none | used to store constraint calculation details |
| | OPT_RTRACE | none | used to store objective function calculations during RSM searches |
| | OPT_RCTRAC | none | used to store constraint calculations during RSM searches |
| | OPT_TSIZE | 1000.0 | |
| | OPT_RESLTS | none | |
| | OPT_CRSLTS | none | |
| | OPT_GRADNT | none | |
| | OPT_NCALLS | none | |
| | OPT_NCCALL | none | |
| response surface methods | FUSION_TYP | 0.0 | |
| | OBJ_THETA | none | Krig hyperparameter |
| | OBJ_EXP | none | Krig hyperparameter |
| | OBJ_LAMBDA | none | Krig hyperparameter |
| | OBJ_BAD_PT | none | |
| | OBJ_BAD_SB | none | |
| | OBJ_CLF | none | Krig concentrated likelihood function |
| | L_OBJ_THET | none | |
| | L_OBJ_EXP | none | |
| | L_OBJ_LAMB | none | |
| | U_OBJ_THET | none | |
| | U_OBJ_EXP | none | |
| | U_OBJ_LAMB | none | |
| | CST_THETA | none | Krig hyperparameter |
| | CST_EXP | none | Krig hyperparameter |
| | CST_LAMBDA | none | Krig hyperparameter |
| | CST_BAD_PT | none | |
| | CST_BAD_SB | none | |
| | CST_CLF | none | Krig concentrated likelihood function |
| | L_CST_THET | none | |
| | L_CST_EXP | none | |
| | L_CST_LAMB | none | |
| | U_CST_THET | none | |
| | U_CST_EXP | none | |
| | U_CST_LAMB | none | |
| | RSM_NSKIP | none | the number of skipped basis fns. for the objective |
| | RSM_NCSKIP | none | the number of skipped basis fns. for the constraints |
| | RSM_NULLH | none | the values of the Null Hypothesis as bases are added for the objective |
| | RSM_CNULLH | none | the values of the Null Hypothesis as bases are added for the constraints |
| | RSM_NULL_T | 10.0 | the % by which the fit quality must decline to halt RBF regression |
| | RSM_SVDTOL | 5.0E-10 | the singular value tolerance in SVD calculations |
| population statistics | | | |

| | | | |
|---|---|---|---|
| | SPM_TRACE | none | |
| | MN_GA | none | mean |
| | SD_GA | none | standard deviation |
| | MS_GA | none | mean + standard deviation |
| | SK_GA | none | skew |
| | KU_GA | none | kutosis |
| | GM_GA | none | geometric mean |
| | GSD_GA | none | geometric standard deviation |
| | MD_GA | none | mode |
| | P05_GA | none | 5% probability level |
| | P50_GA | none | 50% probability level (median) |
| | P95_GA | none | 95% probability level |
| | MIN_GA | none | minimum value |
| | MAX_GA | none | maximum value |
| | MN_EP | none | |
| | SD_EP | none | |
| | MS_EP | none | |
| | SK_EP | none | |
| | KU_EP | none | |
| | GM_EP | none | |
| | GSD_EP | none | |
| | MD_EP | none | |
| | P05_EP | none | |
| | P50_EP | none | |
| | P95_EP | none | |
| | MIN_EP | none | |
| | MAX_EP | none | |
| | MN_ES | none | |
| | SD_ES | none | |
| | MS_ES | none | |
| | SK_ES | none | |
| | KU_ES | none | |
| | GM_ES | none | |
| | GSD_ES | none | |
| | MD_ES | none | |
| | P05_ES | none | |
| | P50_ES | none | |
| | P95_ES | none | |
| | MIN_ES | none | |
| | MAX_ES | none | |
| | MN_PL | none | |
| | SD_PL | none | |
| | MS_PL | none | |
| | SK_PL | none | |
| | KU_PL | none | |
| | GM_PL | none | |
| | GSD_PL | none | |
| | MD_PL | none | |
| | P05_PL | none | |
| | P50_PL | none | |
| | P95_PL | none | |
| | MIN_PL | none | |
| | MAX_PL | none | |
| function plotting | OPT_STEP | 4.0 | only used for auto-scaled plots |
| | WORK | none | used to store plot values |
| BC optimizer | BC_NBIN | 12.0 | |
| | BC_PENAL | 1.0E20 | |

| | | | |
|---|---|---|---|
| | BC_NRANDM | 0.0 | |
| DHC optimizer | DHC_INITSZ | 0.5 | |
| | DHC_THRESH | 0.01 | |
| | DHC_PENAL | 1.0E20 | |
| | DHC_NRANDM | 0.0 | |
| EP optimizer | EP_NBIN | 12.0 | |
| | EP_NPOP | 50.0 | |
| | EP_PENAL | 1.0E20 | |
| | EP_IMUTNT | 2.0 | |
| | EP_TOURN | 0.5 | |
| | EP_NRANDM | 0.0 | |
| | EP_POP | none | |
| ES optimizer | ES_NPPOP | 100.0 | |
| | ES_NCPOP | 1.0 | |
| | ES_PENAL | 1.0E20 | |
| | ES_DELSIG | 0.1 | |
| | ES_UCHILD | 0.0 | |
| | ES_VDSCRT | 1.0 | |
| | ES_MDSCRT | 0.0 | |
| | ES_NRANDM | 0.0 | |
| | EP_PPOP | none | |
| GA optimizer | GA_NBIN | 12.0 | |
| | GA_NPOP | 50.0 | |
| | GA_PENAL | 1.0E20 | |
| | GA_PBEST | 0.8 | |
| | GA_PCROSS | 0.8 | |
| | GA_PINVRT | 0.2 | |
| | GA_PMUTNT | 0.005 | |
| | GA_PRPTNL | 1.0 | |
| | GA_ALPHA | 0.2 | |
| | GA_DMIN | 0.05 | |
| | GA_DMAX | 0.2 | |
| | GA_NCLUST | 0.2 | |
| | GA_NBREED | 0.1 | |
| | GA_PSEED | 0.0 | |
| | GA_NRANDM | 0.0 | |
| | GA_POP | none | |
| | GA_CODE | none | |
| OPTIVAR optimizers | OPT_TOL | 1.0E-5 | TOL in all routines except APPROX, RANDOM & REDUCE |
| | OPT_TOL | 0.001 | G in routine APPROX |
| | OPT_TOL | 0.001 | F in routine RANDOM |
| | OPT_TOL | 0.0001 | CRIT in routine REDUCE |
| | OPT_STEP | 0.02 | 1/NSMAX in routine ADRANS |
| | OPT_STEP | 1.0E-6 | F in routine DAVID |
| | OPT_STEP | 1.0E-6 | F in routine FLETCH |
| | OPT_STEP | 1.0E-6 | F in routine JO |
| | OPT_STEP | 0.1 | STEPMX in routine PDS |
| | OPT_STEP | 0.01 | F in routine SEEK |
| | OPT_STEP | 0.1 | F in routine SIMPLX |
| | OPT_STEP | 0.001 | F in routine APPROX |
| | OPT_STEP | 3.3333E-3 | 1/NSMAX in routine RANDOM |
| | OPT_STEP | 1.0E-6 | EPS in routine REDUCE |
| Misc. optimizers | OPT_TOL | 0.001 | TOL in user supplied routines OPTUM1 and OPTUM2 |
| | OPT_STEP | 1.0E-5 | STEP in user supplied routines OPTUM1 and OPTUM2 |
| | NRP_DX | 1.0E-7 | |
| NAg optimizers | OPT_TOL | 0.001 | |
| | OPT_STEP | $5.0^N$ | |

| | | | |
|---|---|---|---|
| | NAG_ETA | 0.5 | |
| | NAG_RHO | 1.0 | |
| | NAG_BIGBND | 1.0E10 | |
| Num. Rcp. optimizers | OPT_TOL | 0.001 | |
| PBIL optimizer | PL_NBIN | 12.0 | |
| | PL_NPOP | 100.0 | |
| | PL_PENAL | 1.0E20 | |
| | PL_LRATE | 0.05 | |
| | PL_PMUTNT | 0.02 | |
| | PL_NRANDM | 0.0 | |
| | PL_PVECTR | none | |
| DOE optimizer | DOE_NRANDM | 0.0 | |
| | DOE_TRACE | none | |
| | DOE_CTRACE | none | |
| SA optimizer | SA_NBIN | 12.0 | |
| | SA_PTEMP | 0.3333 | |
| | SA_PWIDTH | 5.0 | |
| | SA_PCOLD | 2.0 | |
| | SA_PENAL | 1.0E20 | |
| | SA_PMUTNT | 0.1 | |
| | SA_NRANDM | 0.0 | |
| | SA_SCHED | none | |
| Schwefel optimizers | OPT_TOL | 0.001 | used to control EC, ED, EPSILO(3) and EPSILO(4) † |
| | OPT_STEP | 1.0E-5 | used to control EA, EB, EPSILO(1) and EPSILO(2) † |
| | SC_NRANDM | 0.0 | only used for evolution strategies |
| | SC_LS | 2.0 | |
| | SC_IELTER | 10.0 | |
| | SC_NACHKO | 100.0 | |
| | SC_NS | REAL(NVARS) | |
| | SC_DELS | $1/\sqrt{2N}$ | |
| | SC_DELI | $1/\sqrt{2N/\sqrt{NS}}$ | |
| | SC_DELP | $5 \times 0.01745 = 5°$ | |
| | SC_BKORRL | 1.0 | |
| | SC_KONVKR | 1.0 | |
| † See Chapter 8 for full details. | | | |
| OPTDBS/OPTNS | all the above plus the following: | | |
| | OVR_MAND | 0.0 | turns on mandatory variable bounds in OPTIVR if 1.0 |
| | OVR_PENAL | 1.0 | selects penalty type in OPTIVR routines |
| | OVR_SEED | 128.0 | selects random number seed in ADRANS |
| | OVR_CONV | 1D-4/1D-5 | controls convergence criterion in various OPTIVR routines |
| | OVR_STOP | 0.01 | controls min. step length stopping criterion in SEEK |
| | OVR_STEP | 0.1 | controls fraction of range limiting step lengths in APPROX |
| | OVR_SIMP | 46.0 | controls max. number of simplex iterations in APPROX |
| | OVR_NPTS | 5.0 | controls points retained per iteration in RANDOM |
| | OVR_SHRK | 4.0 | controls shrinkage factor in RANDOM |
| | MC_MAND | 0.0 | turns on mandatory variable bounds in Misc routines if 1.0 |
| | MC_P1 | 0.0D0 | sets first user supplied routine control parameter |
| | MC_P2 | 0.0D0 | sets second user supplied routine control parameter |
| | MC_TYPE | 1.0 | selects search type for DOE and Numerical Recipes methods |
| | MC_PENAL | 1.0 | selects penalty type in Numerical Recipes methods |
| | SC_PENAL | 1.0 | selects penalty type in Schwefel routines |
| | SC_TYPE | 0.0 | selects search type for Powell and MM_ES |
| | SC_CONV | 0.0D0 | controls convergance criterion for DFPS |
| | SC_NITERS | 0.0 | controls number of iterations before convergence checks for Simplex |
| | SC_LR | 1.0 | controls factor on successes per trial for step refinement in 2M_ES |

| | | |
|---|---|---|
| SC_SN | 0.85 | controls step size refinement ratio in 2M_ES |
| SC_IREKOM | 333.0 | controls recombination type in MM_ES |

# APPENDIX M

## Routine & Common Block Names

This appendix details the names of all the routines and common blocks used by the **OPTIONS** package (including those from the Optivar, Schwefel and Numerical Recipes libraries). These names must therefore *be avoided* in user supplied codes, since such useage will prevent the library codes from being correctly linked when building programs. N.b., the NAg and Simpleplot libraries contain many additional routine and common block names which should also be avoided if these libraries are in use - see the documentation supplied with these libraries for further details.

| Name | Name | Name | Name | Name | Name |
|---|---|---|---|---|---|
| ABSCHA | ADDREC | ADRAN | ADRAND | ADRANS | AMONIT |
| ANSWER | ANSWYN | APPROD | APPROX | APPRX | ASKSYS |
| AUXPRT | BLETAL | BRENT | CCKEYS | CGFAM | CHGDIM |
| CHKEYS | CHKMNU | COMP | COMPAR | CON1 | CONECT |
| CONFUN | CONS | CONST | CPRTNM | CPRTSW | CRIT |
| CSTEPM | CSYS | CVSMOD | | | |
| CWSTRG | DASUM | DAT001 | DAT002 | DATAAN | DAVID |
| DAVIDD | DAVIDI | DAXPY | DBRENT | DCONST | DCONTR |
| DCOPY | DDOT | DEFCRV | DEFVS | DEQUAL | DF1DIM |
| DFPMIN | DFPS | DGRAPH | DISURF | DMARK | DMTINV |
| DMTLUB | DMTLUD | DNRM2 | DOT | DSCAL | DSVDCM |
| DSVBKS | DSWAP | DREHNG | DSCG | DSCP | DUREAL |
| DWRANG | E04CGY | EQUAL | EVOL | EVZ | F1COM |
| F1DIM | FC | FD3INT | FF | FFF | FIBO |
| FIBOH | FIND2 | FIND | FLETCD | FLETCH | FLTCH |
| FMIN | FNDWRD | FRANDN | FRPRMN | FUNCT1 | GAUSSN |
| GBLCK1 | GBLCK2 | GBLOCK | GDHELP | GDIN01 | GDIN02 |
| GENBAT | GENCON | GENDAT | GENHID | GENINL | GENINT |
| GET1DN | GET1NO | GETDEF | GETDFD | GETDNM | GETGI |
| GETMNU | GETNUM | GETSTR | GETVI | GETVN | GETVV |
| GETVVD | GETVVS | GIGCTR | GISCTR | GNPOOL | GOLD |
| GOLDH | GRAD | GRSMD | GRSMR | GSTACK | GSTART |
| HILF | HOJE | IM | INCRVD | INDVAL | INDX |
| INTERP | IOCOMM | IODAT | IOLUN | JBDEV | JMY |
| JO | JOB | JOD | KORR | LAGR | LAGRH |
| LINEAR | LINEG | LINEP | LINMIN | LSTGPH | LVNAMS |
| MAIN | MATINV | MATOUT | MINMAX | MNBRAK | MODREC |
| MODTYP | MUTATI | NDSPCH | NOTDIS | NRDFNC | NRFUNC |
| OBJFUN | OPCNT | OPT02 | OPTAMO | OPTBC | OPTBLK |
| OPTCGF | OPTCHK | OPTCLF | | | |
| OPTCNT | OPTCRT | OPTCTL | OPTDBS | OPTDHC | OPTDIM |
| OPTDOE | OPTEP | OPTERR | OPTES | OPTGA | OPTI |
| OPTIM1 | OPTIM2 | OPTIM3 | OPTIM5 | OPTIM | OPTINF |
| OPTIVR | OPTJOB | OPTMON | OPTMSC | OPTMZ1 | OPTMZ2 |
| OPTNAG | OPTNRP | OPTNS | OPTPL | OPTPLT | OPTPLY |
| OPTPNT | OPTPRE | OPTPST | OPTRAD | OPTRES | OPTRSC |

| Name | Name | Name | Name | Name | Name |
|---|---|---|---|---|---|
| OPTRSE | OPTRSF | OPTRSP | OPTRSS | OPTSA | OPTSC1 |
| OPTSCH | OPTSPM | OPTTRC | OPTTRY | OPTUM1 | |
| OPTUM2 | OSWTCH | OTRACE | PACKWD | PALM | PARI |
| PATDUP | PATSON | PDS | PDSD | PDSH | PENDAT |
| PENOUT | PIDATA | PIV | POWE | POWELL | PRTDAT |
| PRTIAL | PRUEFG | QCONST | QEQUAL | QUREAL | RAN |
| RAND | RANDNO | RANDOD | RANDOM | RANLST | RCONST |
| READDN | | | | | |
| READNO | REDUCD | REDUCE | REGRES | RFUNCT | ROSE |
| RRAND | RSTRNG | RTIME | RWRANG | SCONST | SEARCH |
| SEEK | SEEKD | SEEKI | SETDEF | SETDFD | SETGI |
| SETH | SETVV | SETVVD | SFUNCT | SGRAPH | SHOT |
| SIGCTR | SIMP | SIMPL1 | SIMPLD | SIMPLE | SIMPLX |
| SISCTR | SMPLEX | SMPLXD | SOLVE | SPEICH | SRAND |
| SSIMP | SSTACK | START | STRTNG | STIME | TRNSFM |
| TURK1 | TYPDAT | UMSPEI | UPDOT | UPRCAS | UREAL |
| USRDF2 | USRDFN | USRFN2 | USRFTN | WINCLR | WINMNU |
| WRANGE | WSTRNG | Z | ZULASS | | |

# APPENDIX N

# File List for the Standard UNIX Installation

## N.1. Main Directory

This directory holds the principal components of the **OPTIONS** package. It contains the following files:

| | |
|---|---|
| Makefile | - make file for various programs |
| Makefile_intel | - makefile for use with intel compiler |
| Makefile_no_xview | - makefile for use without xview |
| Makefile_pgf | - makefile for use with Portland compiler |
| Makefile_pgf90 | - makefile for use with Portland f90 compiler |
| Makefile_sgi | - makefile for use with SGI compiler |
| Makefile_sun | - makefile for use with Sun compiler |
| beam1con.f | - source for beam1 problem |
| beam1fnc.f | - combined source for beam1 problem |
| beam1fun.f | - source for beam1 problem |
| beam1prg.f | - stand-alone combined source for beam1 problem |
| beam1sc.c | - C source for subroutine based version of beam1 problem |
| beam1s.f | - source for subroutine based version of beam1 problem |
| beam1s.dir | - sub-directory for Windows based solutions to beam1 problem |
| beam2con.f | - source for beam2 problem |
| beam2fun.f | - source for beam2 problem |
| beam2sc.c | - C source for subroutine based version of beam2 problem |
| beam2s.f | - source for subroutine based version of beam2 problem |
| beam3con.f | - source for beam3 multi-objective problem |
| branfun.f | - objective function for Branin problem |
| branin.d | - data-base for Branin function problem |
| bump.c | - description and stand-alone C code for bump problem |
| bump.d | - data file for bump problem |
| bump_2d.d | - data file for two-dimensional version of bump problem |
| bump_1d.d | - data file for one-dimensional version of bump problem |
| bump.dir | - sub-directory for Windows based solutions to bump problem |
| bump.opt | - data-base file for best solution yet found to bump problem |
| bumpcon.f | - source for bump problem |
| bumpfun.f | - source for bump problem |
| cmds | - directory of macro command files |
| cmenu_dummy.c | - dummy routine to replace cmenu routines so that the WIMP interface is not available |
| colv.d | - data file for Colville Function problem |
| colvfun.f | - source for Colville Function problem |
| cont.d | - test 3-D data file for graph utility |
| debcon.f | - constrain function code for Deb multi-objective problem |
| deb.d | - data-base for Deb multi-objective problem |
| designfun.f | - example source file for the control routine of a large design system, see Appendix I |
| doc | - documentation sub-directory (see below) |
| e04cgf.d | - data file for NAg problem |
| e04jaf.d | - data file for NAg problem |
| e04uaf.d | - data file for NAg problem |
| e04ucf.d | - data file for NAg problem |

| | |
|---|---|
| forlib_cdummy.f | - dummy fortran routines for certain functions in Gendat library |
| gdata | - default data file (pointed to by the UNIX environment variable GENDAT_GDATA) |
| gencmd | - macro command file generation utility program |
| gendat.cds | - GENDAT code file (pointed to by the UNIX environment variable GENDAT_CODES) |
| gendat.hlp | - GENDAT help file (pointed to by the UNIX environment variable GENDAT_HELP) |
| gendat_large.f | - routine that should be linked in to allow very large data sets to be used with problems |
| graph.d | - test data file for graph utility |
| graph.f | - source file for graph utility program |
| grid.d | - test 3-D data file for graph utility |
| gstyle | - default graphics style file (pointed to by the UNIX environment variable GENDAT_GSTYLE) |
| idate.f | - a routine for providing date information from the fortran FDATE routine |
| iodat.f | - source for input / output fortan blockdata common block information |
| jhrr.c | - C source for the John Holland Royal Road problem |
| jhrr.d | - data file for the John Holland Royal Road problem |
| jhrr_src | - source directory for the original public domain John Holland Royal Road problem |
| jhrrfun.f | - fortran source for the John Holland Royal Road problem |
| libgendat.a | - main **OPTIONS** and GENDAT library |
| main.c | - source for main calling routine that is used to invoke the XView interface and also parallel job control |
| main_dummy.f | - dummy fortran routine for main function in Gendat library |
| man | - directory of manual source files |
| nagcon.f | - source for NAg problem |
| nagfun.f | - source for NAg problem |
| opdata.d | - skeleton data file |
| opt.f | - source for main package start-up routine |
| optbc_dummy.f | - dummy routine for replacing optbc |
| optblk.f | - source for optimizer fortan blockdata common block information |
| optbump_2d.d | - data-file for use with "optopt" on the bump problem |
| optdhc_dummy.f | - dummy routine for replacing optdhc |
| optep_dummy.f | - dummy routine for replacing optep |
| optes_dummy.f | - dummy routine for replacing optes |
| optga_dummy.f | - dummy routine for replacing optga |
| optivr.d | - data file for OPTIVAR problem |
| optivr_dummy.f | - dummy routine for replacing optivr |
| optivrcon.f | - source for OPTIVAR problem |
| optivrfun.f | - source for OPTIVAR problem |
| optlog_dummy.f | - dummy routine for replacing optlog |
| optmsc_dummy.f | - dummy routine for replacing optmsc |
| optnag_dummy.f | - dummy routine for replacing optnag |
| optnrp_dummy.f | - dummy routine for replacing optnrp |
| optoptfun.f | - source for "optopt" untility program |
| optpl_dummy.f | - dummy routine for replacing optpl |
| optplt_dummy.f | - dummy routine for replacing optplt |
| optpre.f | - skeleton pre-processing routine |
| optpst.f | - skeleton post-processing routine |
| optsa_dummy.f | - dummy routine for replacing optsa |
| optsch_dummy.f | - dummy routine for replacing optsch |
| optsvc.f | - sample web services invocation of **OPTION** |
| optsys.f | - routine used when building optsys |
| optsysfnc.f | - routine used when building optsys |
| optum1.f | - skeleton user defined optimizer one |
| optum2.f | - skeleton user defined optimizer two |
| output | - sub-directory of outputs from various macro command files |
| paretocon.f | - a further Deb multi-objective test case |
| pareto.d | - matching data-base for further Deb multi-objective test case |
| parfun.f | - objective function routine for Pareto optimization |
| parnew.f | - variant of parfun.f that store archive in an external file<br>this enables parallel executions to share a common archive |

| | |
|---|---|
| parnew.fmt | - variant of parnew.f that works with a formatted external file |
| rast.d | - data file for Rastrigin problem |
| rastfnc.f | - source for Rastrigin problem |
| sgraph_dummy.f | - dummy routine for replacing sgraph |
| style | - directory of graphics style files |
| test_all | - linux batch command to test various macro command files |
| tsthst.f | - programme source for host testing facility |
| tstopt.f | - routines to allow codes to run without using Gendat data-base |
| usrftn.f | - source template for user defined functions |
| web | - sub-directory for web page files |
| xview | - sub-directory for linux installation files for xview |

N.b., the dummy files allow various parts of the package to be deleted when compiling if they are not needed. This allows smaller objects to be produced which can help systems with limited memory when running multiple copies of jobs. They are also needed when an installation does not have access to all the components that may be used by the **OPTION** suite.

### N.2. jhrr_src Sub-directory

This sub-directory holds the original, public domain sources for the John Holland Royal Road problem.

| | |
|---|---|
| DESCRIPTION.ps | - postscript description of the problem |
| README | - basic instructions |
| README.holland | - Holland's original posting to the GA electronic digest |
| jhrr.c | - stand-alone C source code used as the basis for "jhrr.c" in the main directory |

### N.3. man Sub-directory

This sub-directory holds the on-line documentation for the **OPTIONS** package. It contains the following files:

| | |
|---|---|
| abstract | - troff source file |
| appdx.a | - troff source file |
| appdx.b | - troff source file |
| appdx.c | - troff source file |
| appdx.d | - troff source file |
| appdx.e | - troff source file |
| appdx.f | - troff source file |
| appdx.g | - troff source file |
| appdx.h | - troff source file |
| appdx.i | - troff source file |
| appdx.j | - troff source file |
| appdx.k | - troff source file |
| appdx.l | - troff source file |
| appdx.m | - troff source file |
| appdx.n | - troff source file |
| bib.dat | - bibliographic data-base file |
| change_history | - summary of changes to package |
| chapter.1 | - troff source file |
| chapter.2 | - troff source file |
| chapter.3 | - troff source file |
| chapter.4 | - troff source file |
| chapter.5 | - troff source file |
| chapter.6 | - troff source file |
| chapter.7 | - troff source file |
| chapter.8 | - troff source file |
| chapter.9 | - troff source file |

| | |
|---|---|
| chapter.10 | - troff source file |
| cont.ps | - contents pages in postscript form from manual |
| figure2.1 | - pic source file |
| figure2.2 | - pic source file |
| figure4.1 | - postscript source file |
| figure4.2 | - pic source file |
| figure4.3 | - pic source file |
| figure5.1 | - pic source file |
| figure5.2 | - pic source file |
| figure5.3 | - pic source file |
| figure6.1 | - postscript source file |
| figure6.2 | - pic source file |
| figure6.3 | - pic source file |
| figure6.4 | - pic source file |
| figure6.5 | - pic source file |
| figure6.6 | - pic source file |
| figure6.7 | - pic source file |
| figure6.8 | - pic source file |
| figure6.9 | - pic source file |
| figure6.10 | - pic source file |
| figure11.1 | - pic source file |
| figureC.1 | - pic source file |
| figureC.10 | - pic source file |
| figureC.11 | - pic source file |
| figureC.12 | - pic source file |
| figureC.2 | - pic source file |
| figureC.3 | - pic source file |
| figureC.4 | - pic source file |
| figureC.5 | - pic source file |
| figureC.6 | - pic source file |
| figureC.7 | - pic source file |
| figureC.8 | - pic source file |
| figureC.9 | - pic source file |
| index | - troff source file |
| index.ps | - index pages in postscript form from manual |
| install.doc | - plain ASCII text installation guide |
| options | - master troff source file |
| options.ps | - postscript version of full manual |
| snapshot.ps | - screen dump postscript file |
| tindex.f | - fortran source for program to concatenate the index to this manual |
| xpic2.1 | - xpic source file |
| xpic2.2 | - xpic source file |

### N.4. cmds Sub-directory

This sub-directory holds the macro command files for the **OPTIONS** package. It contains the following files:

| | |
|---|---|
| beam1.dbs | - builds the beam1 data-base |
| beam1.run | - optimizes the beam1 problem |
| beam2.dbs | - builds the beam2 data-base |
| beam2.run | - optimizes the beam2 problem |
| beam3.dbs | - builds the beam3 data-base and runs a simple Pareto front search |
| beam3.nash | - uses the beam3 data-base to carry out a Nash equilibrium search |
| beam4.dbs | - builds the beam4 data-base |
| beam4.run | - optimizes the beam4 problem |
| bump.run | - optimizes the bump problem |
| bump.pts | - optimizes the bump_2d problem repeatedly to produce a plot showing the scatter |

in solutions

| | |
|---|---|
| bump.rsm_shep | - optimizes the bump_2d problem using Shepard response surface methods |
| bump.rsm_sp | - optimizes the bump_2d problem using stochastic process response surface methods |
| bump.rsm_ei | - optimizes the bump_2d problem using stochastic process response surface methods and the expected improvement operator |
| bump.rsm_clf | - optimizes the bump_2d problem using stochastic process response surface methods and the expected improvement operator, also optimizing the hyperparameters |
| bump.rsm_ply | - optimizes the bump_2d problem using polynomial regression |
| bump.rsm_rbf | - optimizes the bump_2d problem using radial basis functions |
| bump20.rsm_rbf | - optimizes the 20 dimensional bump problem using radial basis functions |
| deb.run | - optimizes the Deb multi-objective Pareto front problem |
| jhrr.run | - optimizes the jhrr problem |
| opt_trace.gr | - example showing how to re-create optimization trace plots from the variable OPT_TRACE |
| optopt.dbs | - adds variables to create the "optopt" data-base |
| optopt.eval | - evaluates the use of all optimizers by using "optopt" |
| optopt.run | - tunes the use of all optimizers by using "optopt" |
| params.dbs | - adds all the variables needed for full control of the various optimizer parameters |
| rsm.dbs | - builds and tunes a RSM even when user has no code, only data which must be keyed in |
| schwefel | - tests all the methods in the Schwefel library on a given problem |
| test.gr | - runs the graph utility test |
| test.3d | - runs the graph utility 3-D test |
| trust | - demonstrates a trust region search on a reduced part of the bump problem |
| tutor | - takes the user through the construction of a simple graph with various prompts and hints |

### N.5. style Sub-directory

This sub-directory holds the graphics style files for the **OPTIONS** package. It contains the following files:

| | |
|---|---|
| bump | - produces the 31 by 31 point bump plot using "bump_2d.d" |
| cont | - produces the non-gridded plots for the graph utility 3-D test |
| graph1 | - produces the linear axes plot for the graph utility test |
| graph2 | - produces the polar axes plot for the graph utility test |
| graph3 | - produces the logarithmic axes plot for the graph utility test |
| graph4 | - produces the histogram plot for the graph utility test |
| graph5 | - produces the cumulative histogram plot for the graph utility test |
| graph6 | - produces the colour polar axes plot for the graph utility test |
| grid | - produces the gridded plots for the graph utility 3-D test |
| opt_ep | - produces the evolutionary programming generation statistics plot |
| opt_es | - produces the evolution strategy generation statistics plot |
| opt_ga | - produces the genetic algorithm generation statistics plot |
| opt_pl | - produces the pop. based inc. learning generation statistics plot |
| opt_iso | - produces iso-metric surfaces from 41x41 point mappings |
| opt_map | - produces contour maps from 41x41 point mappings |
| opt_trace | - plots optimization traces on existing maps or surfaces |
| pareto_front | - produces a Pareto front plot for problems with multiple objectives |

### N.6. doc Sub-directory

This sub-directory holds the documentation files for the libraries used to build the **OPTIONS** package. It contains the following files:

| | |
|---|---|
| READ_ME | - describes the files in the sub-directory |
| exampl.f | - a simple example program for using GENDAT |
| forlib.contents | - lists the contents of the forlib library |
| forlib.hdrs | - subroutine specifications for the forlib library |
| gendat.contents | - lists the contents of the gendat library |
| gendat.hdrs | - subroutine specifications for the gendat library |

| | |
|---|---|
| graph.contents | - lists the contents of the graph library |
| graph.hdrs | - subroutine specifications for the graph library |
| options.contents | - lists the contents of the graph library |
| options.hdrs | - subroutine specifications for the options library |

**N.7. output Sub-directory**

This sub-directory holds the outputs from running a number of the macro command files in the "cmds" sub-directory. It contains the following files all of which stem from the same named files in the "cmds" sub-directory (except where noted):

| | | |
|---|---|---|
| beam1.dbs | | |
| beam1.run | | |
| beam2.dbs | | |
| beam2.run | | |
| beam3.dbs | | |
| beam3.nash | | |
| beam4.dbs | | |
| beam4.run | | |
| bump20.rsm_rbf | | |
| bump.pts | | |
| deb.run | | |
| jhrr.run | | |
| make | - | output from running the linux "make" command using the default Makefile |
| schwefel | | |
| test.3d | | |
| test.gr | | |
| trust | | |

## APPENDIX O

### Listing for programs beam1s.f and beam2s.f

```
C       PROGRAM BEAM1S
        SUBROUTINE WINMNU_PROGRAM(OPTPRG)
C
C         example program that uses the OPTIONS subroutine interface to run the
C         first example in the manual
C
        INTEGER NVRS,NDVRS,NCONS,NPARAM,OLEVEL
        DOUBLE PRECISION DNULL
        PARAMETER(NVRS=2,NDVRS=1,NCONS=5,NPARAM=7,OLEVEL=1,DNULL=-777.0D0)
        LOGICAL ERROR
        INTEGER NITERS,DIRCTN,NCALLS
        DOUBLE PRECISION VARS(NVRS),LVARS(NVRS),UVARS(NVRS)
       1 ,DVARS(NVRS,NDVRS),CONS(NCONS),LCONS(NCONS),UCONS(NCONS)
       2 ,PARAMS(NPARAM),OBJFN,OMETHD
        CHARACTER*(*) OPTPRG
        CHARACTER*10 VARNAM(NVRS),CONNAM(NCONS),PARNAM(NPARAM),OBJNAM
C
C         define the design vector, constraints and objective function
C
C         design vector limits (n.b., to unset a limit set it equal to DNULL)
C
        LVARS(1)=5.0D0
        LVARS(2)=2.0D0
        UVARS(1)=50.0D0
        UVARS(2)=25.0D0
        DVARS(1,1)=DNULL
        DVARS(2,1)=DNULL
        VARNAM(1)='BREADTH'
        VARNAM(2)='HEIGHT'
C
C         constraints vector limits (n.b., to unset a limit set it equal to DNULL)
C
        LCONS(1)=DNULL
        LCONS(2)=DNULL
        LCONS(3)=DNULL
        LCONS(4)=DNULL
        LCONS(5)=5000.0D0
        UCONS(1)=200.0
        UCONS(2)=100.0
        UCONS(3)=5.0D0
        UCONS(4)=10.0D0
        UCONS(5)=DNULL
        CONNAM(1)='SIGMA-B'
        CONNAM(2)='TAU'
        CONNAM(3)='DEFLN'
        CONNAM(4)='H-ON-B'
        CONNAM(5)='F-CRIT'
C
```

```
C          parameters
C
      PARNAM(1)='LENGTH'
      PARNAM(2)='FORCE'
      PARNAM(3)='FACTOR'
      PARNAM(4)='EE'
      PARNAM(5)='GG'
      PARNAM(6)='NU'
      PARNAM(7)='SIGMAY'
      PARAMS(1)=1500.0D0
      PARAMS(2)=5000.0D0
      PARAMS(3)=2.0D0
      PARAMS(4)=216620.0D0
      PARAMS(5)=86650.0D0
      PARAMS(6)=0.27D0
      PARAMS(7)=200.0D0
C
C          objective function
C
      OBJNAM='AREA'
C
C          set up the structure of the problem
C
      PRINT *,'Calling OPTDBS with arguments of '
     1  ,OLEVEL,DNULL,NVRS,NDVRS,NCONS,NPARAM,VARNAM
     2  ,CONNAM,PARNAM,OBJNAM,LVARS,UVARS,DVARS,LCONS,UCONS,PARAMS
      CALL OPTDBS(OLEVEL,DNULL,NVRS,NDVRS,NCONS,NPARAM,VARNAM
     1  ,CONNAM,PARNAM,OBJNAM,LVARS,UVARS,DVARS,LCONS,UCONS,PARAMS,ERROR)
      PRINT *,'Returned from OPTDBS with results of '
     1  ,OLEVEL,DNULL,NVRS,NDVRS,NCONS,NPARAM,VARNAM
     2  ,CONNAM,PARNAM,OBJNAM,LVARS,UVARS,DVARS,LCONS,UCONS,PARAMS,ERROR
      IF(ERROR)THEN
      PRINT *,'Unable to define the problem vectors'
      STOP
      ENDIF
C     CALL OPTCTL
C
C          search the problem using a given method for a given number of steps
C          (method 1.8 is an OPTIVAR Approx search)
C
      OMETHD=1.8
      VARS(1)=30.0
      VARS(2)=7.0
      DIRCTN=-1
      NITERS=100
      PRINT *,' '
      PRINT *,'Calling OPTNS with arguments of '
     1  ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1  ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1  ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to search the problem'
      STOP
      ENDIF
C
```

```
C         search the problem using a given method for a given number of steps
C         (method 4.0 is a GA search)
C
      OMETHD=4.0
      VARS(1)=30.0
      VARS(2)=7.0
      DIRCTN=-1
      NITERS=500
      PRINT *,' '
      PRINT *,'Calling OPTNS with arguments of '
     1 ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1 ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1 ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to search the problem'
      STOP
      ENDIF
C
C         search the problem using a given method for a given number of steps
C         (method 1.6 is an OPTIVAR Seek search)
C
      OMETHD=1.6
      VARS(1)=30.0
      VARS(2)=7.0
      DIRCTN=-1
      NITERS=1000
C
C         set the Fiacco-McCormick penalty
C
      ERROR=.FALSE.
      CALL SETVV('OVR_PENAL',1,1,1,2.0,ERROR)
      IF(ERROR)THEN
      PRINT *,'Failed to set search parameter OVR_PENAL'
      STOP
      ENDIF
      PRINT *,' '
      PRINT *,'Calling OPTNS with arguments of '
     1 ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1 ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1 ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to search the problem'
      STOP
      ENDIF
C
C         just evaluate the problem
C
      OMETHD=0.0
      VARS(1)=30.0
      VARS(2)=7.0
      DIRCTN=-1
      NITERS=1
      PRINT *,' '
```

```
      PRINT *,'Calling OPTNS with arguments of '
     1 ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1 ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1 ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to evaluate the problem'
      STOP
      ENDIF
C
      STOP
      END


C       PROGRAM BEAM2S
      SUBROUTINE WINMNU_PROGRAM(OPTPRG)
C
C       example program that uses the OPTIONS subroutine interface to run the
C       second example in the manual
C
      INTEGER NVRS,NDVRS,NCONS,NPARAM,OLEVEL
      DOUBLE PRECISION DNULL
      PARAMETER(NVRS=3,NDVRS=3,NCONS=5,NPARAM=1,OLEVEL=1,DNULL=-777.0D0)
      LOGICAL ERROR
      INTEGER NITERS,DIRCTN,NCALLS
      DOUBLE PRECISION VARS(NVRS),LVARS(NVRS),UVARS(NVRS)
     1 ,DVARS(NVRS,NDVRS),CONS(NCONS),LCONS(NCONS),UCONS(NCONS)
     2 ,PARAMS(NPARAM),OBJFN,OMETHD
      CHARACTER*(*) OPTPRG
      CHARACTER*10 VARNAM(NVRS),CONNAM(NCONS),PARNAM(NPARAM),OBJNAM
C
C       define the design vector, constraints and objective function
C
C       design vector limits (n.b., to unset a limit set it equal to DNULL)
C
      LVARS(1)=5.0D0
      LVARS(2)=2.0D0
      LVARS(3)=1.0D0
      UVARS(1)=50.0D0
      UVARS(2)=25.0D0
      UVARS(3)=3.0D0
      DVARS(1,1)=DNULL
      DVARS(1,2)=DNULL
      DVARS(1,3)=DNULL
      DVARS(2,1)=DNULL
      DVARS(2,2)=DNULL
      DVARS(2,3)=DNULL
      DVARS(3,1)=1.0D0
      DVARS(3,2)=2.0D0
      DVARS(3,3)=3.0D0
      VARNAM(1)='BREADTH'
      VARNAM(2)='HEIGHT'
      VARNAM(3)='MAT_TYPE'
C
C       constraints vector limits (n.b., to unset a limit set it equal to DNULL)
C
```

```
      LCONS(1)=DNULL
      LCONS(2)=DNULL
      LCONS(3)=DNULL
      LCONS(4)=DNULL
      LCONS(5)=5000.0D0
      UCONS(1)=200.0
      UCONS(2)=100.0
      UCONS(3)=5.0D0
      UCONS(4)=10.0D0
      UCONS(5)=DNULL
      CONNAM(1)='SIGMA-B'
      CONNAM(2)='TAU'
      CONNAM(3)='DEFLN'
      CONNAM(4)='H-ON-B'
      CONNAM(5)='F-CRIT'
C
C       parameters (none in this example - instead rely on GETDEF defaults)
C
      PARAMS(1)=DNULL
      PARNAM(1)=' '
C
C       objective function
C
      OBJNAM='COST'
C
C       set up the structure of the problem
C
      CALL OPTDBS(OLEVEL,DNULL,NVRS,NDVRS,NCONS,NPARAM,VARNAM
     1 ,CONNAM,PARNAM,OBJNAM,LVARS,UVARS,DVARS,LCONS,UCONS,PARAMS,ERROR)
      IF(ERROR)THEN
      PRINT *,'Unable to define the problem vectors'
      STOP
      ENDIF
C     CALL OPTCTL
C
C       search the problem using a given method for a given number of steps
C       (method 1.8 is an OPTIVAR Approx search)
C
      OMETHD=1.8
      VARS(1)=30.0
      VARS(2)=7.0
      VARS(3)=1.0
      DIRCTN=-1
      NITERS=100
      PRINT *,' '
      PRINT *,'Calling OPTNS with arguments of '
     1 ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1 ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1 ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to search the problem'
      STOP
      ENDIF
C
C       search the problem using a given method for a given number of steps
```

```
C         (method 4.0 is a GA search)
C
      OMETHD=4.0
      VARS(1)=30.0
      VARS(2)=7.0
      VARS(3)=1.0
      DIRCTN=-1
      NITERS=500
      PRINT *,' '
      PRINT *,'Calling OPTNS with arguments of '
     1 ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1 ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1 ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to search the problem'
      STOP
      ENDIF
C
C         search the problem using a given method for a given number of steps
C         (method 1.6 is an OPTIVAR Seek search)
C
      OMETHD=1.6
      VARS(1)=30.0
      VARS(2)=7.0
      VARS(3)=1.0
      DIRCTN=-1
      NITERS=1000
C
C         set the Fiacco-McCormick penalty
C
      ERROR=.FALSE.
      CALL SETVV('OVR_PENAL',1,1,1,2.0,ERROR)
      IF(ERROR)THEN
      PRINT *,'Failed to set search parameter OVR_PENAL'
      STOP
      ENDIF
      PRINT *,' '
      PRINT *,'Calling OPTNS with arguments of '
     1 ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1 ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1 ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to search the problem'
      STOP
      ENDIF
C
C         search the problem using a given method for a given number of steps
C         (method 4.0 is a GA search)
C
      OMETHD=4.0
      VARS(1)=30.0
      VARS(2)=7.0
      VARS(3)=1.0
```

```
      DIRCTN=-1
      NITERS=500
      PRINT *,' '
      PRINT *,'Calling OPTNS with arguments of '
     1 ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1 ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1 ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to search the problem'
      STOP
      ENDIF
C
C        search the problem using a given method for a given number of steps
C        but do not reset the starting position
C        (method 1.8 is an OPTIVAR Approx search)
C
      OMETHD=1.8
      DIRCTN=-1
      NITERS=100
C
C        reset the penalty to the internal default
C
      ERROR=.FALSE.
      CALL SETVV('OVR_PENAL',-1,-1,-1,0.0,ERROR)
      IF(ERROR)THEN
      PRINT *,'Failed to reset search parameter OVR_PENAL'
      STOP
      ENDIF
      PRINT *,' '
      PRINT *,'Calling OPTNS with arguments of '
     1 ,OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS,VARS
      CALL OPTNS(OLEVEL,OMETHD,DIRCTN,NITERS,NVRS,NCONS
     1 ,VARS,CONS,OBJFN,NCALLS,ERROR)
      PRINT *,'Returned from OPTNS with results of '
     1 ,VARS,CONS,OBJFN,NCALLS
      IF(ERROR)THEN
      PRINT *,'Failed to search the problem'
      STOP
      ENDIF
C
      STOP
      END
```

# APPENDIX P

## Error Codes

This appendix details the meanings and likely causes of **OPTIONS** failures where numeric error codes are generated.

| Code | Occurs In | Likely Cause |
|---|---|---|
| 0 | Fn. Plotting | Unable to recognize data-base after setting up variable WORK. |
| 1 | Obj. Fn. Calc. | The variables passed to the routine are NaN's or Inf's (unstable optimizer). |
| 2 | | Calc.Unable to recognize data-base after setting up variable OPT_TRACE or OPT_CTRACE. |
| 3 | | Unable to store data in OPT_TRACE or OPT_CTRACE. |
| 4 | | OPTFUN alters data-base structure beyond recognition. |
| 5 | | Failed to add trace point to plot. |
| 6 | | The calculation is beyond single precision. |
| 7 | Equal. Const. Calc. | The variables passed to the routine are NaN's or Inf's (unstable optimizer). |
| 8 | | OPTCON alters data-base structure beyond recognition. |
| 9 | | The calculation is beyond single precision. |
| 10 | | The number of constraints does not tally. |
| 11 | Inequal. Const. Calc. | The variables passed to the routine are NaN's or Inf's (unstable optimizer). |
| 12 | | OPTCON alters data-base structure beyond recognition. |
| 13 | | The calculation is beyond single precision. |
| 14 | | The number of constraints does not tally. |
| 99 | Various | The optimization fails because invalid parameters have been selected. |
| 101 | GENDAT | Invalid key-words used by **OPTIONS** when constructing a menu. |
| 102 | | Insufficient data for numerical integration routine. |
| 103 | | Data not in order for numerical integration routine. |
| 104 | | Menu input returns non-numeric result for numeric input. |
| 105 | | No default option available for constructing menus. |
| 106 | | Invalid data passed for numeric conversion of string input. |
| 107 | | Menu input returns non-numeric result for numeric input. |
| 108 | | Data supplied to numerical integration routine inconsistent (warning). |
| 109 | | Duplicate data supplied to numerical integration routine (warning). |
| 110 | | Invalid matrix size used when trying to carry out inversion. |
| 111 | | Break point data in integration routine not in order. |
| 112 | | Data for numerical integration routine infeasible. |
| 113 | | Convergence not achieved by numerical integration routine (warning). |
| 114 | | Break point data in integration routine not in order. |
| 115 | | Data for numerical integration routine infeasible. |
| 116 | | Convergence not achieved by numerical integration routine (warning). |
| 117 | | Data for numerical regression routine infeasible. |
| 118 | | Contents of macro command file input invalid at this point. |
| 119 | | End of File detected on FORTRAN standard input. |

| Code | Occurs In | Likely Cause |
|---|---|---|
| 120 | GENDAT | Data for numerical interpolation routine infeasible. |
| 121 | | Data for numerical interpolation routine infeasible. |
| 122 | | Data for numerical interpolation routine infeasible. |
| 123 | | Bad auxiliary logical unit number specified by **OPTIONS**. |
| 124 | | Invalid key-words used by **OPTIONS** when constructing a menu. |
| 125 | | Over long strings being checked when comparing names, commands, etc. |
| 126 | | FORTRAN logical unit number invalid for file I/O. |
| 127 | | FORTRAN logical unit number already in use for file I/O. |
| 128 | | Invalid FORTRAN file connection status requested. |
| 129 | | Invalid word list searching task. |
| 130 | | Menu input returns non-numeric result for numeric input. |

# APPENDIX Q

## References


**References**

1. Simpleplot, *Simpleplot Mark 2 Reference Manual,* Bradford University Software Services Ltd., U.K. (1987).
2. J. N. Siddall, *Optimal Engineering Design: Principles and Applications,* Marcel Dekker, Inc., New York (1982).
3. H.-P. Schwefel, *Evolution and Optimum Seeking,* John Wiley & Sons (1995).
4. W.H. Press, B.P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: the Art of Scientific Computing,* Cambridge University Press (1986).
5. NAg E04UCF, *NAg Mark 14 Reference Manual,* Numerical Algorithms Group Ltd., Oxford (1990).
6. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison-Wesley (1989).
7. S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, "Optimization by simulated annealing," *Science* **220**(4598) pp. 671-680 (1983).
8. D. B. Fogel, "Applying evolutionary programming to selected traveling salesman problems," *Cybernetics and Systems* **24**(1) pp. 27-36 (1993).
9. D. Yuret and M. de la Maza, "Dynamic Hill Climbing: Overcoming the Limitations of Optimization Techniques," pp. 254-260 in *Proceedings of the 2nd Turkish Symposium on AI and ANN*, (1993).
10. T. Back, F. Hoffmeister, and H.-P. Schwefel, "A Survey of Evolution Strategies," pp. 2-9 in *Proceedings of the 4th International Conference on Genetic Algorithms (ICGA IV)*, ed. R. K. Belew and L. B. Booker, Morgan Kaufman Publishers, Inc., San Diego (1991).
11. R. B. Statnikov and J. B. Matusov, *Multicriteria Optimization and Engineering,* Chapman and Hall, ISBN 0-412-99231-0 (1995).
12. L. Davis, "Bit-Climbing, Representational Bias, and Test Suite Design," pp. 18-23 in *Proceedings of the 4th International Conference on Genetic Algorithms (ICGA IV)*, ed. R. K. Belew and L. B. Booker, Morgan Kaufman Publishers, Inc., San Diego (1991).
13. S. Baluja, "Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning," CMU-CS-94-163, Carnegie Mellon University (1994).
14. H. Nowacki, "Modelling of Design Decisions for CAD," in *Lecture notes in computer science no. 89: CAD modelling, systems engineering, CAD-systems*, ed. G. Goos, J. Hartmanis, Springer-Verlag, Berlin (1980).
15. NAg E04UAF,UCF, *NAg Mark 12 Reference Manual,* Numerical Algorithms Group Ltd., Oxford (1987).
16. A. J. Keane, "Genetic algorithm optimization of multi-peak problems: studies in convergence and robustness," *Artificial Intelligence in Engineering* **9**(2) pp. 75-83 (1995).
17. M. Mitchell, S. Forrest, and J. Holland, "The Royal Road for Genetic Algorithms: Fitness Landscapes and GA performance," in *Proceedings of the First European Conference on Artificial Life*, MIT Press, Cambridge (1992).
18. J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn, "Design and analysis of computer experiments (with discussion)," *Statistical Science* **4** pp. 409-435 (1989).

19. D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimisation of expensive black-box functions," *J. Global Optimisation* **13**(4) pp. 455-492 (1998).

20. D. Shepard, "A two-dimensional interpolation function for ireegularly spaced data," pp. 517-523 in *Proc. 23rd Nat. Conf. ACM*, , New York (1968).

21. R. Franke and G. Neilson, "Smooth interpolation of large sets of scattered data," *Int. J. Numer. Methods Eng.* **15** pp. 1691-1704 (1980).

22. C. M. Fonseca and P. J. Fleming, "An Overview of Evolutionary Algorithms in Multiobjective Optimization," *Evolutionary Computation* **3**(1) pp. 1-16 (1995).

23. J. D. Knowles and D. W. Corne, "The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Pareto Multiobjective Optimization," in *Proc. Cong. Evolutionary Computation, CEC'99*, IEEE, Piscataway (1999).

24. A. J. Keane, "Statistical Improvement Criteria for Use in Multiobjective Design Optimization," *AIAA Journal* **44**(4) pp. 879-891 (2006).

25. A. V. Fiacco and G. P. McCormick, *Nonlinear programming: sequential unconstrained minimization techniques.,* Wiley (1968).

26. M. J. D. Powell, "A method for nonlinear constraints in minimization problems," in *Optimization*, ed. R. Fletcher, Academic Press (1969).

27. S. B. Schuldt, G. A. Gabriele, R. R. Root, and K. M. Ragsdell, "Application of a new penalty function method to design optimization," *A.S.M.E. Trans., J. Eng. Ind.* **99(B)**(1) pp. 31-36 (1977).

28. R. E. Griffith and R. A. Stewart, "A nonlinear programming technique for the optimization of continuous processing systems," *Management Science* **7** pp. 379-392 (1961).

29. D. S. McArthur, "Strategy in research - alternative methods for design of experiments," *I.R.E. Trans. Eng. Management* **EM-8** pp. 34-40 (1961).

30. D. A. Gall and E. Krokosky, "A generalized procedure for automated optimal design," *Proc. 4th Nat. Conf. on Eng. Design*, (1967).

31. A. Lavi and T. P. Vogl, ","  pp. 369-386 in *Recent advances in optimization techniques*, Wiley (1966).

32. R. Fletcher and M. J. D. Powell, "A rapidly convergent descent method for minimization," *Computer J.* **6** pp. 163-168 (1963).

33. R. Fletcher, "A new approach to variable metric algorithms," *Computer J.* **13** pp. 317-322 (1970).

34. D. H. Jacobson and W. Oksman, "An algorithm that minimizes homogeneous functions of n variables in n+2 iterations and rapidly minimizes general functions," Tech. Report 618, Div. Eng. and Appld. Physics, Harvard University (1970).

35. M. J. D. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *Computer J.* **7**(4) pp. 303-307 (1964).

36. R. Hooke and T. A. Jeeves, *Direct search solution of numerical and statistical problems,* Westinghouse Research Labs. Scientific Paper (1960).

37. J. A. Nelder and R. Meade, "A simplex method for function minimization," *Computer J.* **7** pp. 308-313 (1965).

38. H. H. Rosenbrock, "An automatic method for finding the greatest or least value of a function," *Comp. J.* **3** pp. 175-184 (1960).

39. M. J. Box, "A new method of constrained optimization and a comparison with other methods," *Comp. J.* **8** pp. 42-52 (1965).

40. W. H. Swann, *A report on the development of a new direct searching method of optimization,* ICI Central Instrument Laboratory, Middlesborough (1964).

41. M.R. Anderberg, *Cluster Analysis for Applications,* Academic Press (1975).

42. X. Yin and N. Germay, " A Fast Genetic Algorithm with Sharing Scheme Using Cluster Methods in Multimodal Function Optimization," pp. 450-457 in *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, ed. R. F. Albrecht, C. R. Reeves and N. C. Steele,

Springer-Verlag, Innsbruck (1993).

# Index